CONFINEMENT TUNING OF A 0-D PLASMA DYNAMICS MODEL

A Thesis
Presented to
The Academic Faculty

by

Maxwell D. Hill

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
Department of Nuclear and Radiological Engineering
School of Mechanical Engineering

Georgia Institute of Technology
May 2016

CONFINEMENT TUNING OF A 0-D PLASMA DYNAMICS MODEL

Approved by:

Professor Weston M. Stacey, Advisor
Department of Nuclear and Radiological Engineering
School of Mechanical Engineering
Georgia Institute of Technology

Professor Bojan Petrovich
Department of Nuclear and Radiological Engineering
Georgia Institute of Technology

Dr. Franklin DuBose
Innovative Solutions Unlimited
Project Manager

Date Approved: April 20, 2016

ACKNOWLEDGEMENTS

I wish to thank Dr. Stacey for guiding and directing my research, Tim Collart for helping me to improve my framework for interpreting unusual simulation results, and my wife, Rebecca, for her constant support throughout this research project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

a	Minor radius of the plasma, p. 7.
C_E^e	Electron energy confinement tuning parameter, p. 14.
C_E^i	Ion energy confinement tuning parameter, p. 14.
C_{GAS}	The gas puffing source fitting parameter, p. 10.
C_P^i	Ion particle confinement tuning parameter, p. 14.
C_R	Neutrals recycling parameter, i.e., the fraction of particles leaving the plasma that are assumed to return immediately through reflection and re-emission processes, p. 10.
Δr	The width of each radial mesh in the radial density and temperature distributions, p. 9.
e_e	Charge of the electron, p. 11.
e_i	Charge of the main ion species in the plasma, p. 11.
κ	The plasma elongation, p. 9.
$\langle \sigma v \rangle_f$	Fusion reactivity, p. 6.
$\ln \Lambda$	The Coulomb logarithm, p. 11.
m_e	The mass of the electron, p. 11.
m_i	The mass of the ions collisionally transfering energy with electrons, p. 11.
n_α	Global alpha particle density, p. 6.
n_e	Global electron density, p. 9.
n_i	Global ion density, p. 6.
ν_n	Power to which the parabola in the radial electron density profile is raised, p. 8.
ν_{Te}	Power to which the parabola in the radial electron temperature profile is raised, p. 8.
ν_{Ti}	Power to which the parabola in the radial ion temperature profile is raised, p. 8.
n_z	Global impurity particle density, p. 6.

P^e_{aux}	Auxiliary electron heating power, p. 11.
P^i_{aux}	Auxiliary Ion Heating Power, p. 11.
P_{brem}	Radiated power loss from bremsstrahlung radiation, p. 12.
P_{ECH}	Electron Cyclotron Heating Power, p. 11.
P_{FW}	Fast Wave Heating Power, p. 11.
P_{imp}	Radiated power loss from impurities, p. 12.
P_Ω	Volumetric ohmic heating power, p. 11.
P_R	Radiative power loss, p. 12.
Q_{ie}	Power collisionally transferred from ions to electrons, positive if $T_i > T_e$, p. 11.
r	Radial location in the plasma, p. 7.
R_0	The plasma major radius, p. 9.
ρ	Normalized radial location in the plasma, $\rho \equiv r/a$, p. 7.
ρ_{ped}	Value of ρ at the height of the pedestal, typically $\rho_{ped} \approx 0.9$, p. 7.
r^j	The radial location of the jth radial mesh in the density and temperature profiles, p. 9.
S_α	Volumetric external alpha particle source rate, p. 10.
S_i	Volumetric ion source rate, p. 10.
S_z	Volumetric impurity particle source rate, p. 9.
τ_E^{98}	The ITER-98(y,2) energy confinement scaling law, also written as $\tau_E^{IPB98(y,2)}$, p. 16.
τ_E^e	Electron energy confinement time, p. 6.
τ_E^i	Ion energy confinement time, p. 6.
τ_P^α	Alpha particle confinement time, p. 6.
τ_P^i	Ion (and electron) particle confinement time, p. 6.
τ_P^z	Impurity particle confinement time, p. 6.
T_e	Global electron temperature, p. 6.
T_i	Global ion temperature, p. 6.

$U_{\alpha e}$ Energy deposited to electrons from fusion α-particles, p. 11.

U_i The energy of the ions collisionally transfering energy with electrons, taken to be $3/2T_i$, p. 11.

$U_{\alpha i}$ Energy deposited to ions from fusion α-particles, p. 11.

V^j The volume of the toroidal shell corresponding to the jth radial mesh, p. 9.

W_b The energy of the neutral beam as it thermalizes in the plasma. Initially $W_b \approx 80kev$ for DIII-D shots., p. 12.

SUMMARY

Fusion power excursions resulting from the strong positive temperature dependence of the D-T fusion rate would have significant consequences for heat removal in ITER and subsequent tokamaks. While several active power excursion control mechanisms have been and continue to be investigated, the impending operation of ITER provides a strong incentive to investigate the possibility of physical mechanisms and configurations that would inherently limit incipient power excursions. To this end, a computationally efficient predictive model has been developed that adjusts confinement characteristics based on results from regression analyses of DIII-D data.

While the ITER-98 law represents a correlation of data from a wide range of tokamaks, confinement scaling laws will need to be fine tuned to specific operational features of specific tokamaks in the future. A methodology for developing, by regression analysis, tokamak- and configuration-specific confinement tuning models is presented and applied to DIII-D as an illustration. It is shown that inclusion of tuning parameters in the confinement models can significantly enhance the agreement between simulated and experimental temperatures relative to simulations in which only the ITER-98 scaling law is used. These confinement tuning parameters can also be used to represent the effects of various heating sources and other plasma operating parameters on overall plasma performance and may be used in future studies to inform the selection of plasma configurations that are more robust against power excursions.

CHAPTER I

BACKGROUND

1.1 The Promise of Fusion

Nuclear fusion, the process by which stars create vast amounts of energy, holds great promise as an energy source for later in the 21st century. In fusion, light elements are accelerated toward each other with enough energy to overcome the coulomb repulsive force and allow the strong nuclear force to merge the two atoms. The result is a new atom and the release of a large amount of energy and other small particles.[1]

Throughout the 20th century, the most successful fusion reactor design has been the tokamak, which confines plasma in an doughnut-shaped magentic bottle. A diagram of the DIII-D tokamak, from which the data used in this work was obtained, is shown in Figure 1. The overall performance of tokamak fusion reactors, as measured by the Lawson Criteria defined in Equation 1,[1] has increased steadily over the last several decades and has even outpaced Moore's Law, which predicts that the number of transistors per square inch on integrated circuits will double every year, as shown in Figure 2. Many efforts across the fusion community are currently being directed toward the ITER tokamak in France, which will demonstrate the physics and engineering requirements necessary for the successful operation of future fusion power reactors.

$$Lawson\ Criteria = n_i T_i \tau_e \tag{1}$$

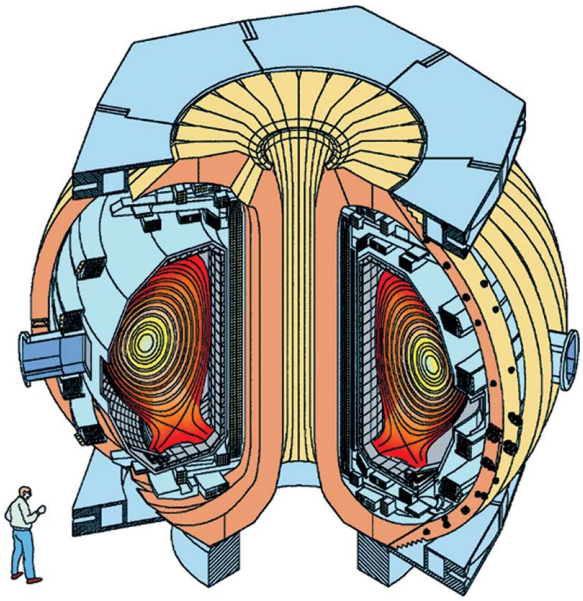


Figure 1: Diagram of the DIII-D tokamak.

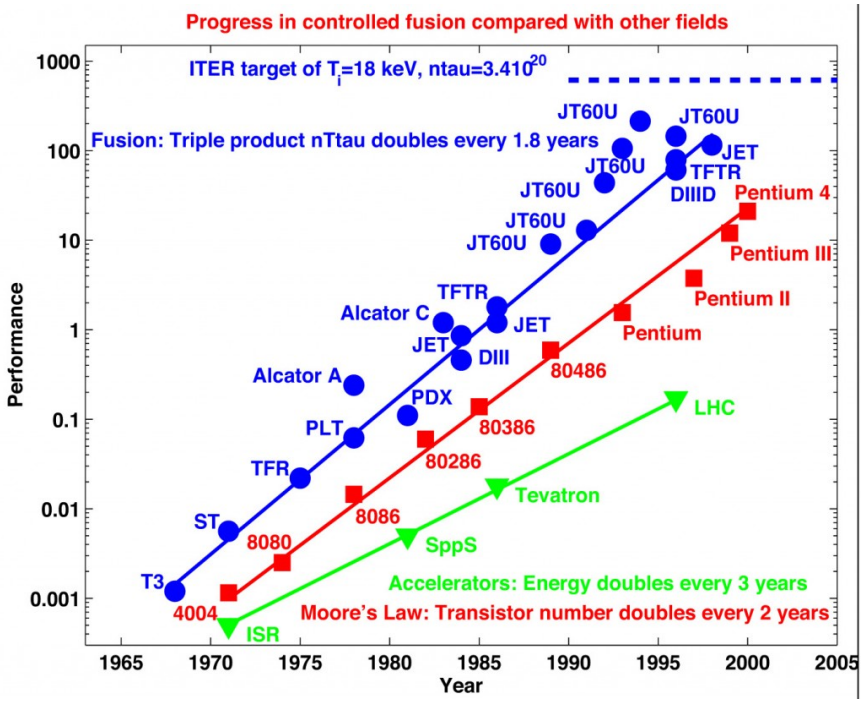


Figure 2: Comparison of Lawson Criteria and Moore's Law over the last several decades.

1.2 Power Excursions

ITER and subsequent power reactors are expected to generate power by fusing approximately equal amounts of deuterium and tritium (D-T fusion), which has a much larger fusion cross section than does deuterium fusing with itself (D-D fusion), as shown in Figure 3. However, almost all experimental tokamak research up to this point has used deuterium-only plasmas due to the significant challenges associated with the production and storage of tritium, as well as because deuterium plasmas allow the investigation of relevant plasma physics phenomena with less damage to the fusion reactors.

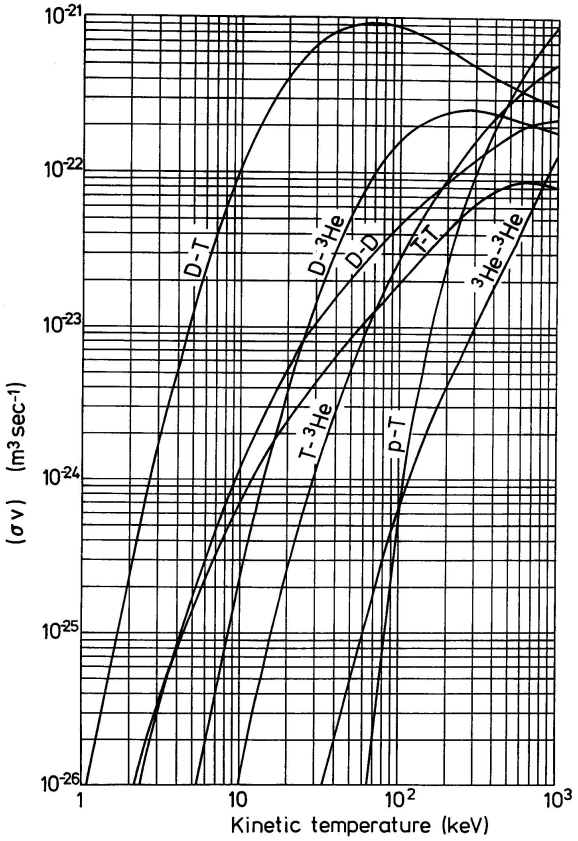


Figure 3: Fusion reactivities for several reactions as a function of temperature.

The fusion cross-section for D-T plasmas has a very strong positive temperature dependence in the range of temperatures that future fusion reactors are expected to operate in. It has long been recognized[2, 3, 4, 5, 6, 7, 8] that the strong positive

temperature dependence of the D-T fusion rate raises the issue of positive power excursions in a fusion power reactor. Accidental changes in plasma operating parameters (e.g. activation of an inactive plasma heating source) can also potentially cause power excursions. Power excursions of even 10 or 20% would have significant consequences for heat removal in ITER[9] and could damage key tokamak components either directly or indirectly by driving a disruptive or other instability.[1] The results of a fusion power excursion in larger reactors or in the fusion neutron sources of future fusion-fission hybrids,[10] such as the Subcritical Advanced Burner Reactor (SABR) concept being developed at Georgia Tech,[11] could be significantly more deleterious.

1.3 Burn Control and Plasma Simulation

Much of the early burn control research was conducted using simple 0-D particle and power balance models to investigate control mechanisms that could be used to mitigate power excursions. There has since been an enormous improvement in the ability to calculate transport and stability in the plasma[12, 13, 14, 15, 16] and there has been much progress in using fitted experimental results to model plasma parameters.[17, 18, 19] Several sophisticated codes have also been developed for the dynamic modeling and simulation of various tokamak plasma scenarios. Notably, the Tokamak Simulation Code (TSC)[20] and the FASTRAN code[21] have been successful in modeling transport, heating, current drive, MHD equilibrium, and stability of a variety of shots. Most recently, an adaptive nonlinear control algorithm capable of actively stabilizing plasma parameters in a burning tokamak plasma was also developed[22] using a 0-D model similar to the one described in this thesis.

Active power excursion control mechanisms that have been investigated include variation of the fueling or heating rate,[23] decompression of the plasma,[24] the injection of impurities,[25] and others.[26, 27, 3, 28, 29] However, the identification of physical mechanisms that would inherently limit incipient power excursions is of

particular importance. For example, operating conditions in which confinement scales negatively with temperature or positively with radiation, soft beta limits, etc. would limit any power excursion without requiring active control action. Although some such mechanisms have been investigated in the past,[28] the impending operation of ITER provides a strong incentive to revisit this issue.

1.4 Passive Stability and the GTBURN Code

The overall objective of the present research project, of which the work reported in this thesis is the first part, is to identify inherent feedback mechanisms and/or control actions that can mitigate the effects of a positive fusion power excursion in ITER and subsequent tokamaks. The methodology for this research will draw on methodology that has been developed for simulating and controlling present experiments as well as "burn control" methodology that has been developed to control power excursions in D-T plasmas.

The first step in this larger effort is to develop a computationally economical model for global plasma dynamics simulation and "train" it to better model experimental results. To this end, a global model has been developed and trained to predict density and temperature evolution in several different types of DIII-D discharges for comparison with measured quantities and to explore the effects of changes in inputs and other operating parameters on the dynamic evolution of plasma conditions. The methodology used to train this model to predict DIII-D dynamics could be used to train a similar model to predict the dynamics of ITER, once it becomes operational. This thesis reports the development, by regression analysis of DIII-D data, of confinement tuning parameters that refine the ITER-98 scaling law[30] to better simulate the temperatures of DIII-D[31] plasmas under a variety of operating conditions.

CHAPTER II

PHYSICS MODEL

2.1 Main Equations

A code, GTBURN, has been developed to solve the following global non-linear 0-D equations that govern burning plasma performance.[1] All particle and power sources in these equations are volume-averaged quantities.

$$\frac{dn_i}{dt} = S_i - \frac{1}{2}n_i^2 \langle\sigma v\rangle_f - \frac{n_i}{\tau_P^i} \tag{2}$$

$$\frac{dn_\alpha}{dt} = S_\alpha + \frac{1}{4}n_i^2 \langle\sigma v\rangle_f - \frac{n_\alpha}{\tau_P^\alpha} \tag{3}$$

$$\frac{dn_z}{dt} = S_z - \frac{n_z}{\tau_P^z} \tag{4}$$

$$\frac{3}{2}\frac{d}{dt}(n_e T_e) = P_\Omega + P_{aux}^e + \frac{1}{4}n_i^2 \langle\sigma v\rangle_f U_\alpha^e - Q_{ie} - P_R - \frac{3}{2}\frac{n_e T_e}{\tau_E^e} \tag{5}$$

$$\frac{3}{2}\frac{d}{dt}(n_i T_i) = P_{aux}^i + \frac{1}{4}n_i^2 \langle\sigma v\rangle_f U_\alpha^i + Q_{ie} - \frac{3}{2}\frac{n_i T_i}{\tau_E^i} \tag{6}$$

Here, n_i, n_α, and n_z, are the volume-weighted deuterium, α-particle, and impurity densities, respectively. Because the primary goal in this thesis is to model the DIII-D plasma, the tritium and alpha-particle densities are not modeled. When benchmarking against DIII-D shots, it is assumed that the ion is deuterium. T_e and T_i are, respectively, the average electron and ion temperatures that have been weighted by both density and volume. The fusion reactivity, $\langle\sigma v\rangle_f$, is shown in Figure 3 and is discussed more in Appendix A.2.

The quantities τ_P^i, τ_P^α, τ_P^z, τ_E^e, and τ_E^i, are the ion, α-particle, and impurity particle confinement times and electron and ion energy confinement times, respectively. Each

are assumed to scale with the ITER-98(y,2) scaling law,[30] i.e., $\tau_P^i = C_P^i \times \tau_E^{98}$. The C multipliers are confinement-tuning parameters (CTPs) used to refine the ITER-98 scaling law to better account for the effects of specific operating parameters and control actions (e.g. types of heating power, particle source rates, changes in geometry, etc.), on energy and particle confinement in a specific tokamak. The objective of this thesis is to determine these CTPs using measured DIII-D data and construct a predictive model for DIII-D from them using regression analysis.

2.2 Density and Temperature Averaging

Volume-averaged ion densities are used in the global model and are calculated from synthesized radial density distributions. Profiles are modeled as a "parabola to a power on a pedestal" in the core and as a straight line in the edge pedestal region, as described in Equation 7.

$$n_e(\rho) = \begin{cases} [n_e(0) - n_e(\rho_{ped})]\left(1 - \frac{\rho^2}{\rho_{ped}^2}\right)^{\nu_n} + n_e(\rho_{ped}) & 0 \le \rho \le \rho_{ped} \\ \\ \frac{n_e(\rho_{ped})}{1-\rho_{ped}}(1-\rho) & \rho_{ped} \le \rho \le 1 \end{cases} \tag{7}$$

Here, $\rho \equiv r/a$, where r is the radial position in the plasma and a is the minor radius of the plasma. The value of ρ at the height of the edge pedestal is denoted as ρ_{ped}, and is approximately 0.9 in most shots. A comparison of this radial profile model with radial data from DIII-D is shown in Figure 4. Temperature distributions are modeled in the same manner.

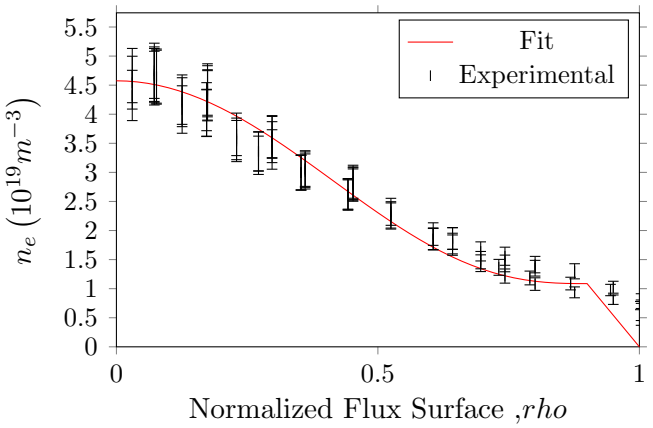


Figure 4: A comparison of the "parabola to a power on a pedestal" fit with radial ion temperature data for DIII-D shot 134350.

The powers ν_n, ν_{Ti}, and ν_{Te} to which the parabolas for the electron density, ion temperature, and electron temperature, respectively, are raised are calculated at periodic intervals throughout the evolution of the shot using measured values of those quantities at $\rho = 0$, $\rho = 0.5$, and $\rho = \rho_{ped}$. The value of ν can be obtained by solving for it in Equation 7 and using the density or temperature value at any third point in the radial distribution. For our analysis, we have chosen to use the value at $\rho = 0.5$.

$$\nu_n = \frac{\ln\left[\frac{n_e(0.5)-n_e\left(\rho_{ped}\right)}{n_e(0)-n_e\left(\rho_{ped}\right)}\right]}{\ln\left[1-\left(\frac{0.5}{\rho_{ped}}\right)^2\right]} \tag{8}$$

When interpolating between density and temperature values at different times, GTBURN first interpolates between the values of the radial profile at $\rho = 0$, $\rho = 0.5$, and $\rho = 0.9$ input by the user and calculates a new value of ν at each timestep to better capture the effects of changes in the shape of the various radial distributions throughout the shot.

Volume-weighted densities are calculated by discretizing radial profiles into J radial meshes and averaging as shown in Equation 9.

$$\overline{n_e}\left(t\right) = \frac{\sum_{j=1}^{J} V^j\left(t\right) n_e^j\left(t\right)}{\sum_{j=1}^{J} V^j\left(t\right)} \tag{9}$$

Here $V^j(t)$ is the volume of each toroidal shell for a torus with an elliptical cross section and a radial mesh size Δr.

$$V^j\left(t\right) = 2\pi^2 \kappa R_0 \left(r^j \Delta r + \left(\Delta r\right)^2\right) \tag{10}$$

where κ is the plasma elongation, R_0 is the plasma major radius, and r^j is the radius of the jth radial mesh. Global ion and electron temperatures are are weighted both by volume and also by the radial density distribution.

$$\overline{T_e}\left(t\right) = \frac{\sum_{j=1}^{J} V^j\left(t\right) n_e^j\left(t\right) T_e^j\left(t\right)}{\sum_{j=1}^{J} V^j\left(t\right) n_e^j\left(t\right)} \tag{11}$$

2.3 Particle Balance

Although the model can include any number of impurity species, this analysis considers only a single effective intrinic carbon species with a density and representative radial impurity density profile based on results available from the ONETWO code[12] for the DIII-D shots considered in this thesis. For this work, total impurity fractions are treated as fixed and the impurity source S_z, which is primarily the result of physical sputtering of the plasma-facing components, is not explicitly modeled. Future sputtering calculations will be performed as described in Appendix A.1. The ion density, n_i, radial profile is calculated from the measured electron (n_e) and impurity

radial density profiles and the assumption of charge neutrality. A volume-averaged electron density is then calculated as described in Equation 9. Because the α-particle density is neglible in DIII-D, S_α, n_α, and τ_P^α are not explicitly modelled in this work.

The ion source S_i is modeled as the sum of contributions from neutral beam injection (NBI), gas puffing, and recycled neutrals from the wall, as described in Equations 12 through 15. C_{GAS} is a scaling parameter. The gas puffing source was scaled to provide physically meaningful results and account for the fact that not all gas puffing particles actually enter the plasma. A value of C_{GAS}= 150 was found to work well.

$$S_i\left(t\right) = S_{i,NBI} + S_{i,gas} + S_{i,recycled} \tag{12}$$

$$S_{i,NBI}\left(t\right) = \frac{P_{NBI}}{eW_b} \tag{13}$$

$$S_{i,gas}\left(t\right) = \frac{GAS\left(Torr \cdot L/s\right) N_A}{R_{GAS} C_{GAS}} \tag{14}$$

$$S_{i,rec}\left(t\right) = C_R \frac{n_i}{\tau_P^i} = C_R \frac{n_i}{C_P^i \tau_E^{98}} \tag{15}$$

Both reflection and delayed re-emission of neutrals are challenging effects to accurately model because they depend on plasma conditions in previous discharges, wall conditioning, and many other considerations. A detailed investigation of these phenomena is beyond the scope of this work, but has been explored in other studies.[32, 33, 34] The neutrals recycling model used in this thesis assumes that a fixed fraction, C_R, of the ions that leave the plasma are immediately reflected and return to the plasma. The calculation of C_R is discussed in the next chapter.

2.4 Power Balance

2.4.1 Power Balance Inputs

The rate of ion-electron collisional energy transfer Q_{ie} is calculated from[1]

$$Q_{i,e}\left(\frac{W}{m^3}\right) = n_i \frac{dU}{dt} = \frac{n_i n_e \left(e_i e_e\right)^2 m_e \ln\Lambda \left(-\frac{2U_i}{3T_e}\right)}{2\pi\epsilon_0 \left(2\pi m_e T_e\right)^{1/2} m_i \left(1 + \frac{4\sqrt{\pi}}{3}\left(\frac{m_e U_i}{m_i T_e}\right)^{3/2}\right)} \tag{16}$$

where e_i, e_e, $\ln\Lambda$, U_i, m_e, and m_i are the charges of the ion and electron, the coulomb logarithm, the energy of the ion, and the masses of the electron and ion respectively. Q_{ie} is calculated using a radial distribution of collisional energy transfer rates is calculated using radial density and temperature distributions, which is then weighted by volume and averaged to calculate a global Q_{ie}.

$U_{\alpha e}$ and $U_{\alpha i}$ are the energies transferred from fusion α-particles to the electron and ion populations, respectively. Although fusion alpha-heating is negligible in the D-D shots considered in this thesis, these terms have been included for future use.

Ohmic heating (P_Ω), electron cyclotron heating (P_{ECH}) and the amount of fast wave power coupled to the plasma (P_{FW}) are taken from DIII-D data and volumetrically averaged. In this model, it is assumed that P_{ECH} is deposited entirely to the electron population and that P_{FW} is deposited to the ions. The total auxiliary heating to the electrons, P^e_{aux}, consists of P_{ECH} and the portion of the NBI heating power deposited to the electrons. Similarly, P^i_{aux} consists of P_{FW} and the portion of NBI power deposited to the ions.

2.4.2 Neutral Beam Energy Deposition Calculation

NBI heating power is also taken from DIII-D data and is distributed between ions and electrons for each DIII-D shot according to calculations of the instantaneous power deposited in the plasma ion and electron populations as the fast ions become thermalized in the plasma. The instantaneous energy transfer to each population is

calculated according to Equations 17 and 18.[1]

$$\frac{dW_b^{ion}}{dt} = -\frac{2^{\frac{1}{2}} n_e Z^2 Z_b^2 e^4 M_b^{\frac{1}{2}} \ln\Lambda}{8\pi\epsilon_0^2 m_i W_b^{\frac{1}{2}}} \tag{17}$$

$$\frac{dW_b^{electron}}{dt} = -\frac{2^{\frac{1}{2}} n_e Z_b^2 e^4 M_e^{\frac{1}{2}} \ln\Lambda}{6\pi^{\frac{3}{2}} \epsilon_0^2 M_b T_e^{\frac{3}{2}}} W_b \tag{18}$$

The beam energy, W_b, is then reduced by the amount of energy lost and the calculation is performed again. This process is repeated until $W_b \leq T_e$, at which point the beam is said to be thermalized. This process occurs on a much faster timescale than the general evolution of densities and temperatures in the plasma and is assumed to occur instantaneously in the model. This calculation is performed at every timestep in the code. The contributions to the ions and electrons throughout this process are then summed at each timestep to obtain the total fraction of power deposited to the ion and electron populations by the beams at that instant.

In reality, this process happens over a range of densities and temperatures as the beam particles move through the plasma. For simplicity, this process is assumed to happen at a single density and temperature that are representative of core conditions where a majority of the beam heating power is known to be deposited.[1] Improvements to the model to more accurately treat beam energy deposition will be made in the future. Heating from fusion alpha particles is negligible in the shots considered in this thesis, however the ion and electron deposition ratios from fusion alpha particles can, in principle, be calculated using a similar approach.

The radiative power loss density, P_R, is evaluated for impurity (P_{imp}) and bremsstrahlung (P_{brem}) radiation losses and is calculated from [1].

$$P_{imp}\left(\frac{MW}{m^3}\right) = (1 + 0.3T_e) \times 10^{-43} n_e n_z z^{(3.7 - 0.33 \ln T_i)} \tag{19}$$

$$P_{brem}\left(\frac{W}{m^3}\right) = 1.7 \times 10^{-38} z^2 n_i n_e T_e^{\frac{1}{2}} \tag{20}$$

CHAPTER III

TUNING PARAMETER MODEL

3.1 DIII-D Shot Selection

The shots considered for model development and validation were limited to ELMing H-mode shots (non-RMP) with $B_0 < 0$ (i.e. the magnetic field in the counter-current direction, which is the standard configuration in most DIII-D shots). The modeling of large MHD events has not been attempted and shots were excluded from consideration if they contained such events. The shots used to develop the models for the confinement multipliers and the ranges of some key parameters for those shots are listed in Table 1. Here, the ranges of P_{ECH}, P_{FW}, and P_{NBI} are the total (rather than volumetric, as is used in the physics model) electron-cyclotron, FW, and neutral beam heating powers, respectively. The ranges of several machine and operating parameters across all selected shots are shown in Table 2.

3.2 Confinement Tuning Parameters

Equations 2 through 6 can be readily solved using a forward finite difference approximation and initial conditions determined from experiment. A time step of 0.01 seconds was used for all simulations described in this thesis. When experimental densities, temperatures, and machine and operating parameters are known, Equations 2 through 6 can be solved "backwards" for the implied particle and energy confinement times, $\tau_{P,E}^{i,e}$, at all times during the evolution of the shot. In our case, we are interested in modeling the confinement as the product of the IPB98(y,2) scaling law and a confinement tuning parameter. The tuning parameters C_P^i, C_E^i, and C_E^e, are defined

Table 1: Characteristics of each shot used in this study. Each shot is an ELM-ing H-mode shot. Starred shots are validation shots that were not used to develop the confinement tuning parameter models.

Shot	$P_{ECH}\,(MW)$	$P_{FW}\,(kW)$	$GAS\,(Torr \cdot L/s)$	$P_{NBI}\,(MW)$	$\lvert B_0\rvert\,(T)$	q_{95}	$n_i\,(10^{19}m^{-3})$	$T_e\,(keV)$
131190*	0-2.41	0	0	2.98-8.38	1.75-1.85	4.85-5.25	2.55-3.44	1.63-2.96
131191	0-2.36	0	0	2.95-8.05	1.74-1.85	4.81-5.23	2.16-2.77	1.46-3.23
131195	0	0	0	3.03-4.35	1.81-1.85	4.79-5.05	2.27-2.57	1.65-2.04
131196	0-1.24	0	0	4.14-9.32	1.79-1.82	4.85-5.12	2.15-2.64	1.87-2.68
134350	0-3.12	0	0-63.91	2.83-9.14	1.75-1.89	5.01-5.75	2.05-2.50	1.46-3.26
135837	0	0	0-43.19	1.35-14.44	1.74-1.99	7.11-7.99	2.62-4.55	0.57-1.79
135843	0	1.69-1.86	0-0.10	3.51-4.40	1.89-1.95	3.14-3.19	5.98-6.76	1.38-1.60
140417	0	0	0	2.31-4.37	1.90-1.97	4.73-4.85	2.60-3.52	1.24-1.62
140418*	0	0	0	4.12	1.92-1.95	4.63-4.76	3.40-4.11	1.18-1.42
140419	0	0	0	4.12	1.93-1.98	4.62-4.88	3.18-3.84	1.22-1.43
140420*	0-3.22	0	0	4.12	1.95-1.99	3.85-4.06	3.31-5.11	1.33-1.78
140421	0-3.18	0	0	4.12	1.94-1.99	4.19-4.40	2.62-3.88	1.27-1.93
140422	0	0	0	4.12	1.93-1.98	4.22-4.42	3.21-3.79	1.38-1.68
140423	0	0	0	4.12	1.93-2.02	4.21-4.44	2.44-3.51	1.43-1.74
140424	0	0	0.33-90.61	4.12	1.94-1.98	4.21-4.35	3.21-5.66	0.91-1.61
140425	0	0	0-108.36	4.12	1.94-1.98	4.18-4.32	4.03-5.56	0.72-1.41
140427*	0	0	0-108.79	4.12	1.94-2.01	4.15-4.47	4.86-5.61	0.86-1.12
140428	0	0	0	4.12	1.96-2.01	4.17-4.32	4.28-5.98	0.90-1.11
140429	0	0	0	4.12	1.95-1.99	4.16-4.31	3.81-5.52	0.98-1.18
140430	0	0-19.11	0	4.12	1.95-2.00	4.53-4.61	2.93-3.40	1.39-1.56
140431	0	0-20.60	0	4.12	1.95-2.01	4.68-4.80	2.62-3.21	1.41-1.71
140432	0	0	0	4.12	1.94-1.99	4.75-4.85	2.61-2.92	1.76-1.96
140440	0	0	0-108.46	3.13-4.14	1.95-1.99	4.58-4.79	4.21-6.29	0.69-1.40
140535*	0	2.46-2.70	0	2.41-4.46	2.02-2.07	4.12-4.30	1.18-1.58	1.43-1.78
140673	0.89-3.32	0-229.95	0	6.89-11.05	1.67-1.70	5.75-6.02	2.85-3.71	1.29-1.59

*Shots not used in model construction, but for validation purposes

Table 2: Range of parameters for DIII-D shots in this study

Parameter	**Range**
$P_{NBI}\,(MW)$	$1.04 - 14.44$
$P_{NBI,counter}/P_{NBI,total}$	$0.0 - 0.45$
$P_{ECH}\,(MW)$	$0.0 - 3.32$
$P_{FW}\,(MW)$	$0.0 - 0.23$
q_{95}	$3.14 - 8.05$
$I_P\,(MA)$	$0.57 - 1.48$
$\lvert B_0\rvert\,(T)$	$1.67 - 2.08$
$S_{i,GAS}\,(10^{22}s^{-1})$	$0 - 7.03$
$\delta_{divertor}$	$0.04 - 0.72$
$\delta_{non-divertor}$	$0.05 - 0.7$
κ	$1.5 - 1.88$
$n_{i,av}\,(10^{19}m^{-3})$	$1.15 - 6.74$
$T_{i,av}\,(keV)$	$0.52 - 3.07$
$T_{e,av}\,(keV)$	$0.47 - 3.41$

as

$$C_{P,E}^{i,e}(t) = \frac{\tau_{P,E}^{i,e}(t)}{\tau_E^{IPB98(y,2)(t)}} = \frac{\tau_{P,E}^{i,e}}{0.056 I_P^{0.93} |B_0^{0.15}| \left(\frac{n}{10^{20}}\right)^{0.41} M^{0.19} R^{1.97} \kappa^{0.78} A^{0.58} P^{-0.69}} \quad (21)$$

where τ_E^{98} is the ITER-98(y,2) energy confinement scaling law shown below.

$$\tau_E^{98} = 0.056 \frac{I_P^{0.93} |B_0^{0.15}| \left(\frac{n}{10^{20}}\right)^{0.41} M^{0.19} R^{1.97} \kappa^{0.78} A^{0.58}}{P^{0.69}} \quad (22)$$

Here, $A = R/a$, $P\,(MW)$ is the total heating power, $I_P\,(MA)$ is the total plasma current, and κ is the plasma elongation.

As described previously, volume-averaged densities are calculated at regular intervals throughout the shot. "Experimental" time traces for the averaged densities and temperatures are obtained by linearly interpolating between values at several points in the shot. Figure 5 illustrates the evolution of the density profile in DIII-D Shot 140673 and the resulting evolution of the averaged quantities.

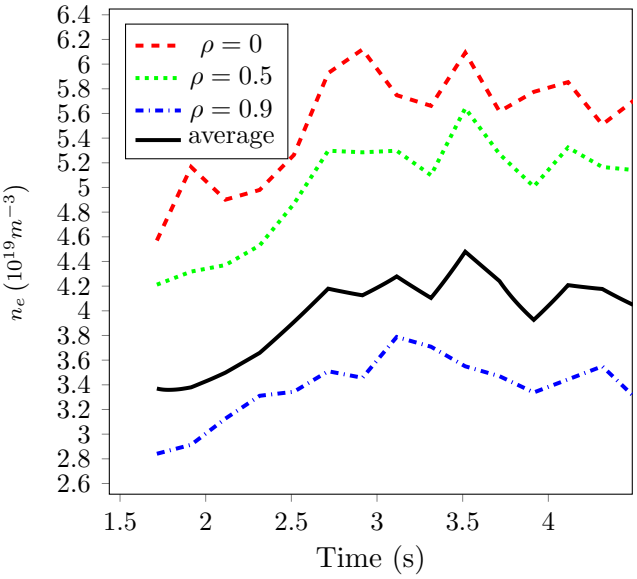
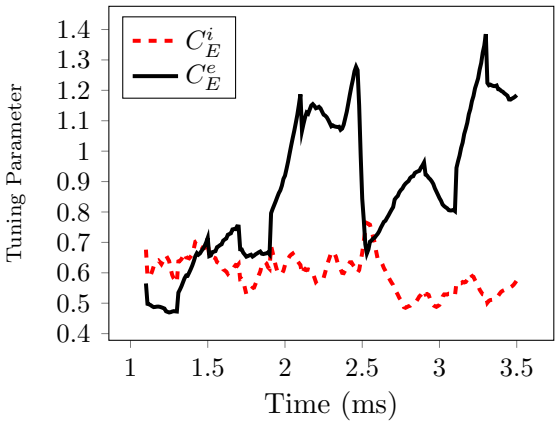
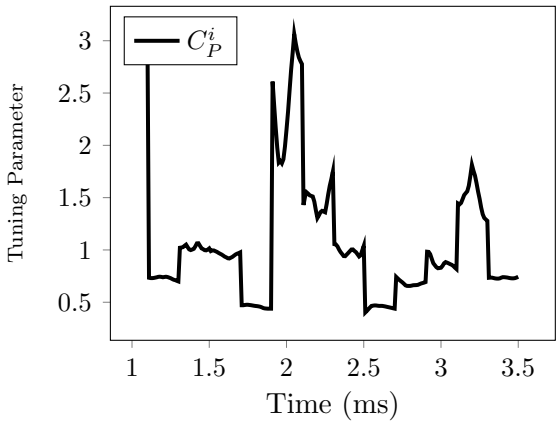


Figure 5: Time traces of the evolution of temperature and density at different radial locations and the resulting averaged quantities for shot 140673.

The "experimental" density and temperature time traces allow us to solve Equations 2 through 6 for the implied confinement tuning parameters throughout the shot. The evolution of the various tuning parameters is shown for several DIII-D shots in Figures 6 through 8.

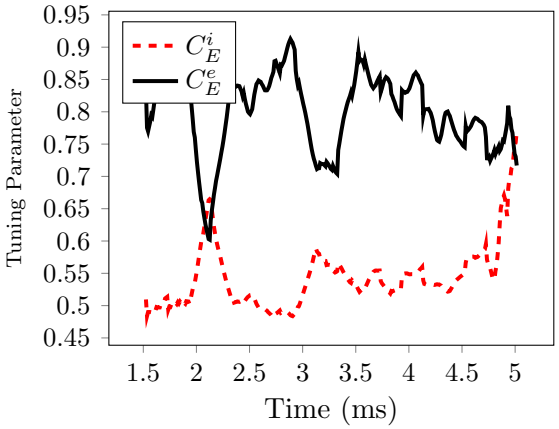


(a) Evolution of the ion and electron energy confinement tuning parameters.

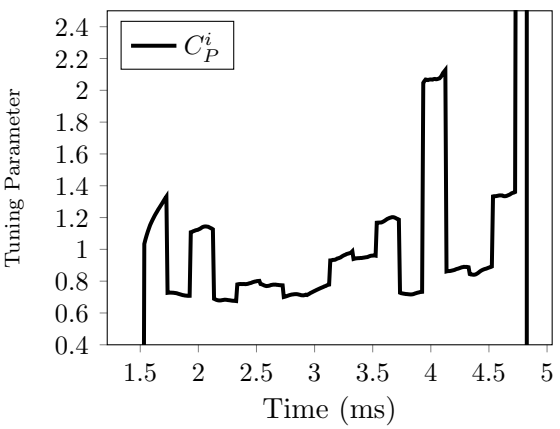


(b) Evolution of the particle confinement tuning parameter.

Figure 6: Time trace of the confinement tuning parameters for shot 131191.

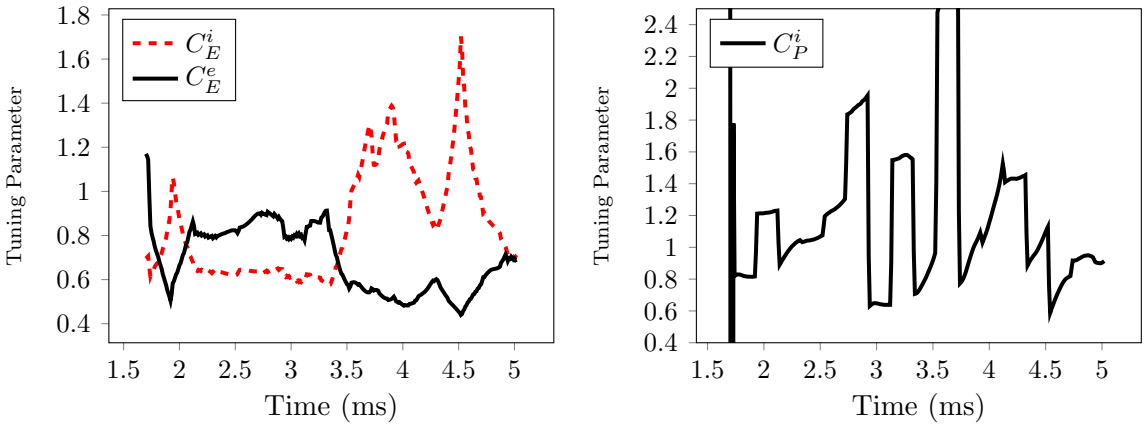


(a) Evolution of the ion and electron energy confinement tuning parameters.



(b) Evolution of the particle confinement tuning parameter.

Figure 7: Time trace of the inferred confinement tuning parameters for shot 140417.



(a) Evolution of the ion and electron energy confinement tuning parameters.

(b) Evolution of the particle confinement tuning parameter.

Figure 8: Time trace of the inferred confinement tuning parameters for shot 140422.

Predictive models for the tuning parameters were constructed using multiple regression analysis on data taken at fixed intervals from the non-asterisked 20 shots shown in Table 1. The predictive variables that were selected for use in the models and the form of the resulting regression models are shown in Equation 23. Table 3 shows the regression parameters for the density and temperature confinement models that were constructed using the shots listed in Table 1.

$$\begin{aligned} C = b_1 + b_2 P_{NBI,vol} + b_3 P_{NBI,counter-frac} + b_4 P_{NBI,short-frac} + \\ b_5 P_{ECH,vol} + b_6 P_{FW,volumetric} + b_7 q_{95} + b_8 q_0 + b_9 I_{P,xsec} + \\ b_{10} B_{T,0} + b_{11} S_{i,GAS,vol} + b_{12} \delta_{divertor} + b_{13} \delta_{non-divertor} + b_{14} \kappa \end{aligned} \quad (23)$$

Here, $I_{P,xsec}$ is the average current density in the plasma, i.e., $I_P / (\kappa a^2)$. $P_{NBI,counter-frac}$ is the fraction of the total NBI power that was injected counter-current and $P_{NBI,tan-frac}$ is the fraction of the total NBI power traveling along the four most tangential paths, shown as solid lines in Figure 9. For the standard DIII-D I and B directions shown in Figure 9, the 210° beams are injected counter-current and the other beams are

injected co-current. These independent variables were considered in order to compensate for the simplistic beam energy deposition model currently used in the code.

By modeling both the reflected and re-emitted neutral sources as a fixed fraction C_R of the number of particles leaving the plasma, these sources can be combined into a single recycled source model, and the global ion particle balance can be rewritten as shown in Equation 24.

$$\frac{dn_i}{dt} = S_{i,NBI} + S_{i,gas} - (1 - C_R)\frac{n_i}{C_P^i \tau_e^{98}} - \frac{1}{2} n_i^2 \langle \sigma v \rangle_f \tag{24}$$

Because of significant uncertainty in both the re-emitted source and the particle confinement time in Equation 24, the value of C_R was determined by assuming that $C_P^i = 1$ (i.e. that $\tau_P^i = \tau_E^{98}$), solving for the the implied recycled source rate at fixed intervals throughout all shots, and regressing those source rates against the particle transport loss rates. The result of this regression is a value of $C_R = 0.904$. With this model for the recycled neutrals source, it was found that none of the independent parameters used to construct regression models for the other tuning parameters (i.e. $P_{ECH,vol}$, $S_{i,GAS,vol}$, etc.) had significant additional explanatory power. As such, the particle confinement tuning parameter is set to unity and not modified based on changes in plasma conditions or operating parameters.

The regression coefficients for the energy confinement tuning models as well as the statistical R^2 values (denoted as R^2_{stats} to avoid confusion with the plasma major radius) and p-values from the regression analysis are shown in Table 3 (p-values less than about 0.05 are generally considered to be statistically significant). Based on the reported p-values and iterating over several model variations, it appears that the operating and machine parameters that have the most effect on the ion energy CTP are the FW input power, plasma elongation, the volumetric neutral beam heating power, and the value of q_0. For the electron energy CTP, it appears that the most

important factors are the absolute value of the toroidal magnetic field, the value of q_0, the volumetric ECH and FW power inputs. The negative regression coefficient for ECH in the electron energy confinement model could also be interpreted as accounting for less than perfect ECH efficiency. Although less important overall, it is interesting to note that ion energy confinement appeared to depend more sensitively on the triangularity in the divertor region ($\delta_{divertor}$) while the electron energy confinement depends more sentitively on the non-divertor triangularity ($\delta_{non-divertor}$).

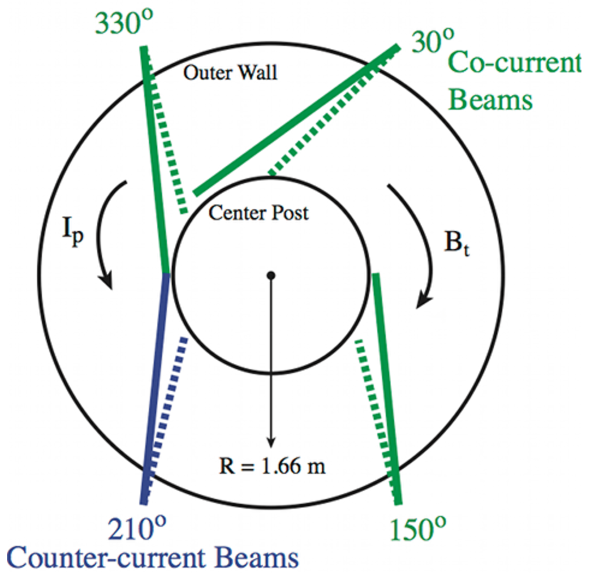


Figure 9: NBI injection angles from the 8 sources on DIII-D, as seen from above

Table 3: Regression Coefficients for Confinement Tuning Models and associated p-values

	Corresponding Parameter	C_E^i	C_E^i p-value	C_E^e	C_E^e p-value
b_1		5.167	1.72×10^{-33}	4.425	2.51×10^{-23}
b_2	$P_{NBI,vol}$	-0.604	3.12×10^{-45}	-0.654	1.09×10^{-48}
b_3	$P_{NBI,counter-frac}$	0.072	5.48×10^{-06}	-0.216	1.64×10^{-38}
b_4	$P_{NBI,short-frac}$	0.056	1.56×10^{-02}	-0.203	6.01×10^{-17}
b_5	$P_{ECH,vol}$	-0.154	1.01×10^{-06}	-0.919	3.90×10^{-160}
b_6	$P_{FW,volumetric}$	35.722	1.19×10^{-89}	-49.129	2.00×10^{-150}
b_7	q_{95}	0.068	5.17×10^{-06}	0.197	3.51×10^{-36}
b_8	q_0	-0.088	2.25×10^{-40}	-0.203	2.50×10^{-179}
b_9	$I_{P,xsec}$	1.237	2.82×10^{-12}	2.829	3.48×10^{-52}
b_{10}	$B_{T,0}$	-0.402	1.67×10^{-06}	-3.095	1.80×10^{-243}
b_{11}	$S_{i,GAS,vol}$	-0.041	1.65×10^{-36}	-0.041	1.51×10^{-34}
b_{12}	$\delta_{divertor}$	0.178	6.06×10^{-20}	-0.104	2.17×10^{-07}
b_{13}	$\delta_{non-divertor}$	0.325	1.90×10^{-12}	-0.675	3.06×10^{-44}
b_{14}	κ	-2.594	2.80×10^{-72}	0.301	4.14×10^{-2}
R^2_{stats}		0.482		0.541	

Many of the operating parameters in a tokamak are correlated with many other parameters. Strong correlations between parameters in a regression model, such as the one used in this project, can be problematic because they can obscure the relative importance of the independent variables used in the regression. A correlation matrix was also computed for the 13 operating parameters used in the construction of the CTP models. This matrix, shown in Table 4, shows strong correlations between $P_{NBI,short-frac}$, $P_{NBI,vol}$, q_{95}, and $B_{T,0}$. These correlations are most likely the result of both the limited number of shots used in this study as well as the importance of all of them in achieving certain desirable modes of confinement. Strong correlations can also be seen between $B_{T,0}$, κ, and $\delta_{non-divertor}$. These correlations are not surprising as parameters that describe the shape of the plasma (including κ and $\delta_{non-divertor}$) are strongly affected by the strength of the toroidal field. Despite the correlations between them, these parameters can capture different important physical effects and should not be excluded from the model because of their relatively high level of correlation.

Finally and unsurprisingly, a strong positive correlation is observed between $I_{P,xsec}$ and q_{95}, although both were shown to be statistically significant. Because of this correlation and the fact that q_{95} was not as important as other independent parameters in explaining the changes in the CTPs, the removal of q_{95} from the regression model would most likely not be problematic. This will be explored in future studies.

Table 4: Correlation Matrix for Independent Variables Used in Regression Analysis

	$P_{NBI,vol}$	$P_{NBI,counter-frac}$	$P_{NBI,short-frac}$	$P_{ECH,vol}$	$P_{FW,volumetric}$	q_{95}	q_0	$I_{P,xsec}$	$B_{T,0}$	$S_{i,GAS,vol}$	$\delta_{divertor}$	$\delta_{non-divertor}$	κ
$P_{NBI,vol}$	1.00												
$P_{NBI,counter-frac}$	0.21	1.00											
$P_{NBI,short-frac}$	-0.89	-0.15	1.00										
$P_{ECH,vol}$	0.37	-0.12	-0.45	1.00									
$P_{FW,volumetric}$	0.37	-0.08	-0.39	0.12	1.00								
q_{95}	0.75	0.21	-0.76	0.09	0.27	1.00							
q_0	0.22	0.20	-0.27	-0.12	0.05	0.48	1.00						
$I_{P,xsec}$	-0.54	-0.12	0.57	0.04	-0.43	-0.81	-0.30	1.00					
$B_{T,0}$	-0.73	-0.04	0.76	-0.38	-0.45	-0.66	-0.52	0.53	1.00				
$S_{i,GAS,vol}$	-0.09	-0.06	0.06	-0.16	-0.07	-0.07	0.04	0.03	0.15	1.00			
$\delta_{divertor}$	0.44	0.02	-0.48	0.08	0.22	0.58	0.42	-0.16	-0.52	-0.14	1.00		
$\delta_{non-divertor}$	0.58	0.06	-0.66	0.32	0.27	0.64	0.53	-0.30	-0.80	-0.13	0.81	1.00	
κ	0.35	0.00	-0.42	0.31	0.06	0.42	0.51	-0.25	-0.77	-0.13	0.44	0.61	1.00

3.3 Interpretation of Confinement Tuning Parameters

3.3.1 General Interpretation

Because the confinement tuning parameters are merely fitting parameters used to adjust the model predict experimental results, it can sometimes be difficult to properly interpret the physical meaning of changes in the tuning parameters. For example, a higher value of the tuning parameter may not coincide with a higher value of the implied confinement time because changes in machine and operating parameters may simultaneously cause a decrease in the ITER-98 scaling law and an increase in the confinement tuning parameter. Similarly, it is possible for a tuning parameter to increase while the corresponding plasma attribute (i.e. density or temperature) decreases due to other simultaneous effects.

The confinement tuning parameters are best thought of as an indication of how a change in an operating or machine parameter is likely to change particle or energy confinement relative to the ITER-98 scaling law, all else being equal. They can capture the dependence of plasma performance on parameters that are not included in the ITER-98 scaling law like, for example, the presence of fast wave current drive or the percentage of power that is injected in teh counter-current direction.

The "segmented" appearance of some tuning parameter time plots, such as the one shown in Figure 10, is typically the result of a coarse time resolution of the input data, between which GTBURN linearly interpolates. If, for example, the given density data is at 200 ms time intervals and the heating power, plasma current, and other inputs are given at much smaller intervals, there is a high likelyhood that the interpolated density data will become inconsistent with experimental values during that time window. This results in periodic corrections in the value of the tuning parameter. Because of limitations in the DIII-D software, obtaining radial density and temperature data at small time intervals is prohibitvely time-consuming, however tools may exist that can provide this data more efficiently. This will be pursued in the future.

3.3.2 Negative and other non-physical tuning parameters

Because the confinement tuning parameters are first calculated using experimental data, errors in those data can result in strange behavior in the evolution of the confinement tuning parameter. These errors could include misallocating input power to the ions and electrons, over- or under-estimating the particle source from recycled neutrals, errors in the values of densities and temperatures from interpolation over too large a time interval, etc. An example of the erratic behavior in the tuning parameter that can result is shown for the ion energy confinement tuning parameter in DIII-D shot 140427 in Figure 10. In this shot, the erratic behavior coincides with strong gas

puffing, which is consistent with a general observation that strong gas puffing tends to degrade the ability to model energy confinement tuning parameters, even when gas puffing is included in the regression model. This phenomenon will be investigated in future studies.

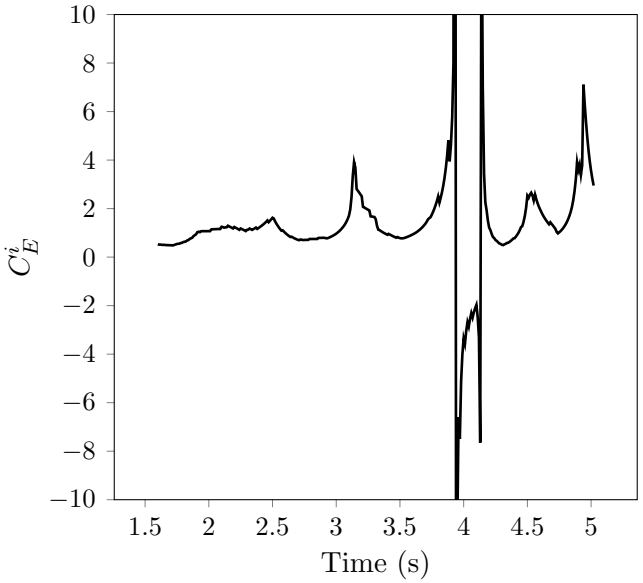


Figure 10: Erratic behavior of the ion energy confinement tuning parameter C_E^i during DIII-D shot 140427.

Although the sharp peaks and occasionally strange behavior may appear to be similar to a numerical instability, they are actually the code adjusting the confinement tuning parameters to compensate for errors in either the experimental data or in calculated parameters elsewhere in the model. Occasional sharp peaks or negative values can be attributed to uncertainty in experimental data or calculated parameters and are unlikely to significantly alter the analysis or the insights derived from that analysis. Longer periods of erratic or non-physical behavior may indicate a problem with the physics model.

By adjusting the confinement tuning parameter, GTBURN adjusts the entire confinement loss term to balance the various particle and power balance equations. This fact allows the user to gain insight into potential problems or weaknesses in

the physics model. A qualitative plot of the confinement tuning parameter vs. the magnitude of the loss term is shown in Figure 11. This relationship holds for all confinement tuning parameters and the transport loss terms they appear in. Four regions are shown that correspond to different interpretations. Those interpretations are described in Table 5.

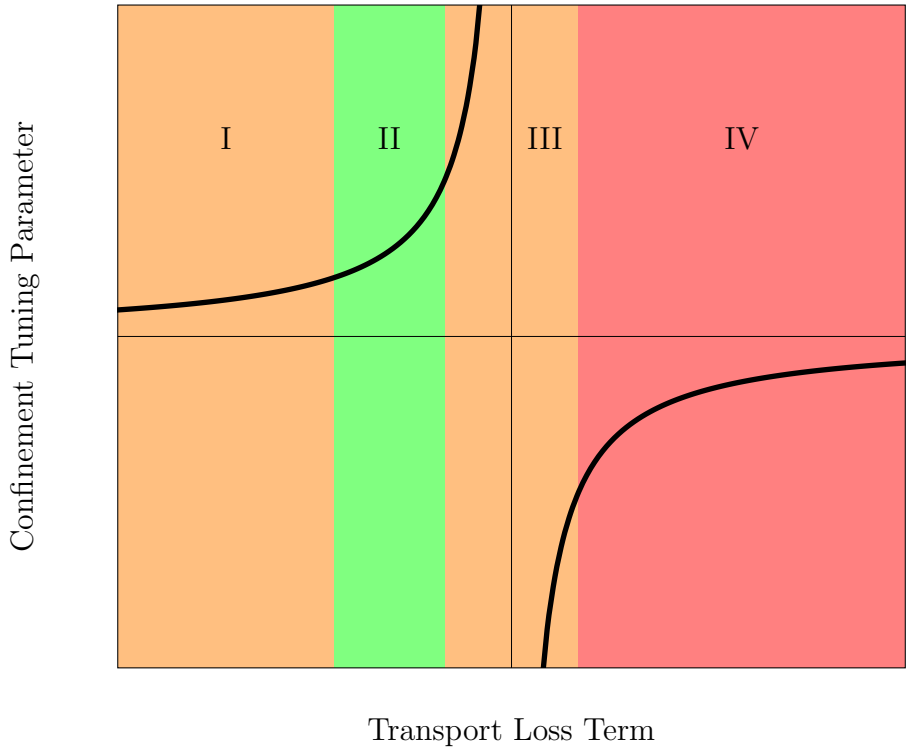


Figure 11: Qualitative plot of the relationship between confinement tuning parameters and the transport loss terms they appear in. Four regions are identified and described in Table 5.

Insight can also be gained by comparing the ion and electron energy confinement tuning parameters. If the behaviors of the tuning parameters and their corresponding interpretations suggest a missing source in one and too much of a source in the other, there may be problems in, for example, the beam energy deposition model or the interspecies collisional energy transfer calculation. If problematic tuning parameter behavior is present in only one of the tuning parameters, it suggests that there may be problems in inputs that primarily effect either the ions or the electrons.

Table 5: Interpretation of Regions in Figure 11

Region	**Tuning Parameter is:**	**Interpretation**
I	Small and positive	The tuning parameter has become very small and positive, resulting in a very large loss term in the balance equation. This indicates a missing sink or a source that is too large.
II	Mid-range and positive	This is the ideal region in which the tuning parameters are typically between about 0.5 and 1.5. In this region, the transport loss term is negative and approximately the correct magnitude. Meaningful tuning parameter dynamics can easily be discerned.
III	Large and positive or large and negative	A large value in the denominator (positive or negative) results in a small loss or source term. Since we expect a moderate loss term, this indicates a missing source or a sink that is too large, but less strongly than results in Region I.
IV	Small and negative	In order to balance, the tuning parameter has had to go negative and small, turning the transport loss term into a large source term. This indicates that the tuning parameter is compensating for a missing large source or the presence a sink that is too large.

CHAPTER IV

COMPARISON WITH EXPERIMENT

4.1 Overview

The confinement tuning regression coefficients of Table 3 were used in predictive simulations of densities and temperatures in 20 DIII-D shots upon which they were based as well as five additional shots (the asterisked shots in Table 1), which served as validation shots. To determine the improvement resulting from the use of the confinement tuning parameters, each simulation was made with and without the use of confinement tuning parameters. These simulations are compared with experiment in the figures below as CTP and No CTP, respectively.

Shots used to build and validate the model were H-mode, single-node divertor shots with the toroidal magnetic field in the counter-current direction, as shown in Figure 9. Shots with large MHD phenomena were not included as the model does not attempt to simulate such phenomena. Shots were selected to include a variety of changes in control parameters like power input and gas puffing in order to investigate the validity of the model in a dynamic environment. Much of the data were obtained from a collisionality scan that included both moderate and near-zero triangularity shots, which provided additional diversity to the dataset

Difficulties were encountered in accurately modeling changes in density. These challenges were likely due in part to the recycled neutrals model, which will be improved in the future. In order to test the construction of the energy confinement tuning parameters, the densities in Figures 12 through 16 have been set to the experimental values.

4.2 Validation Shot Simulations

The first validation shot, shown in Figure 12, is an ELMing H-mode shot that is more or less in steady-state throughout the shot. This shot had no gas puffing, a relatively small triangularity of about 0.07, and an elongation of 1.75. The simulation showed excellent quantitative agreement with the measured electron temperature. Although simulated ion temperatures were approximately 15% greater than experiment throughout the simulation, the results show excellent qualitative agreement with experiment.

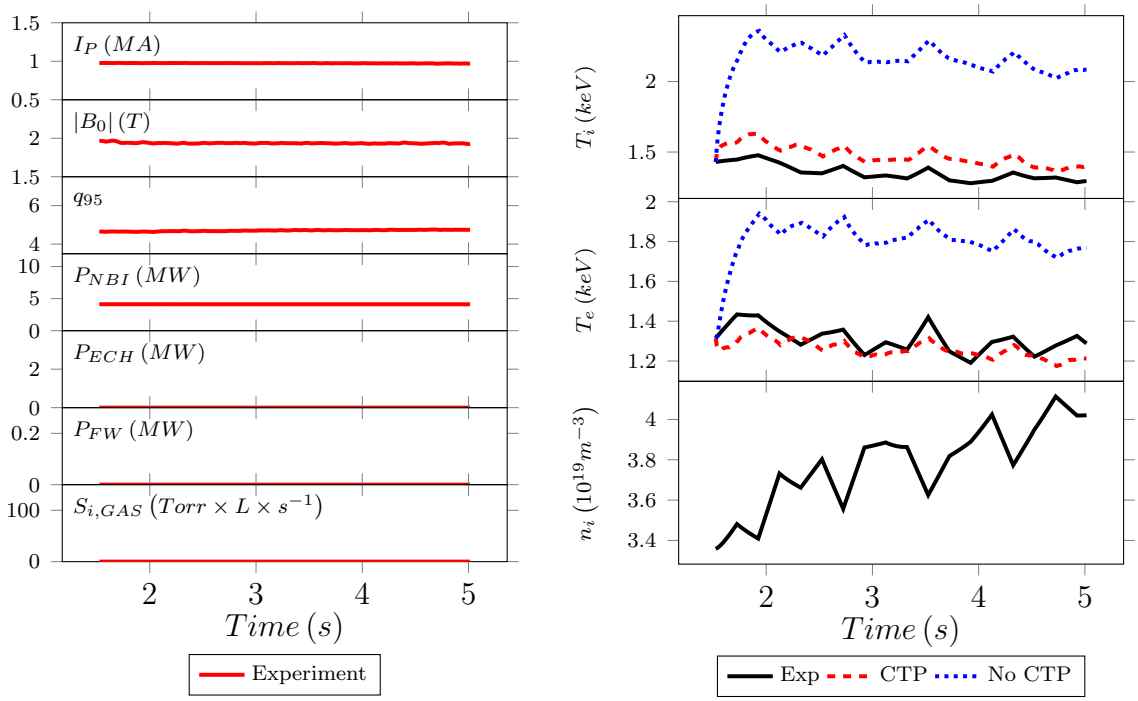


Figure 12: Comparison of simulated and experimental temperatures for DIII-D validation shot 140418, which is very nearly steady-state. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

Temperature simulations for the second validation shot, shown in Figure 13, is also an ELM-ing H-mode shot. This simulation exhibited generally good agreement with experiment in response to ECH being turned on in the middle of the shot followed by a gradual increase in NBI power. Both the ion and electron temperature variations matched the experimental results quite well.

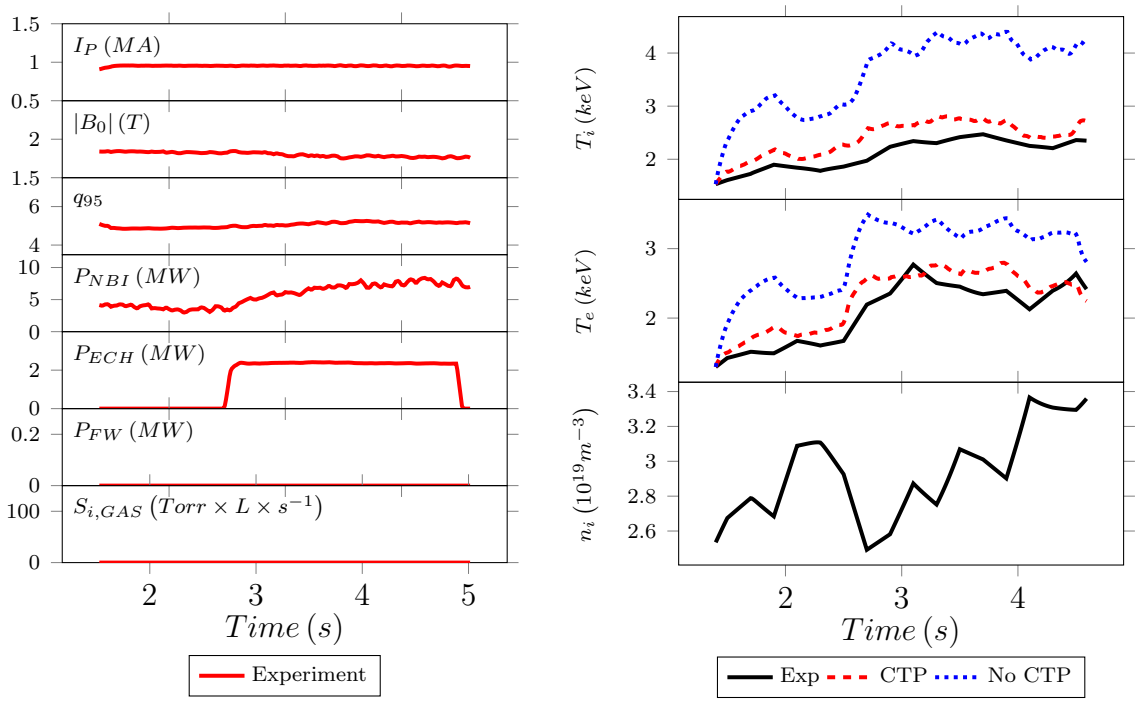


Figure 13: Comparison of simulated and experimental temperatures for DIII-D shot with variable NBI and ECH (131190).

The third validation shot, shown in Figure 14, contained NBI heating ranging from 2.4 - 4.4 MW as well as FW and density changes. Quantitative agreement was worse on this shot than most others, however the simulation still shows reasonably good qualitative agreement with experiment. The opposite behaviors of the ion and electron temperature simulations may suggest the need for an improvement in the neutral beam power deposition model.

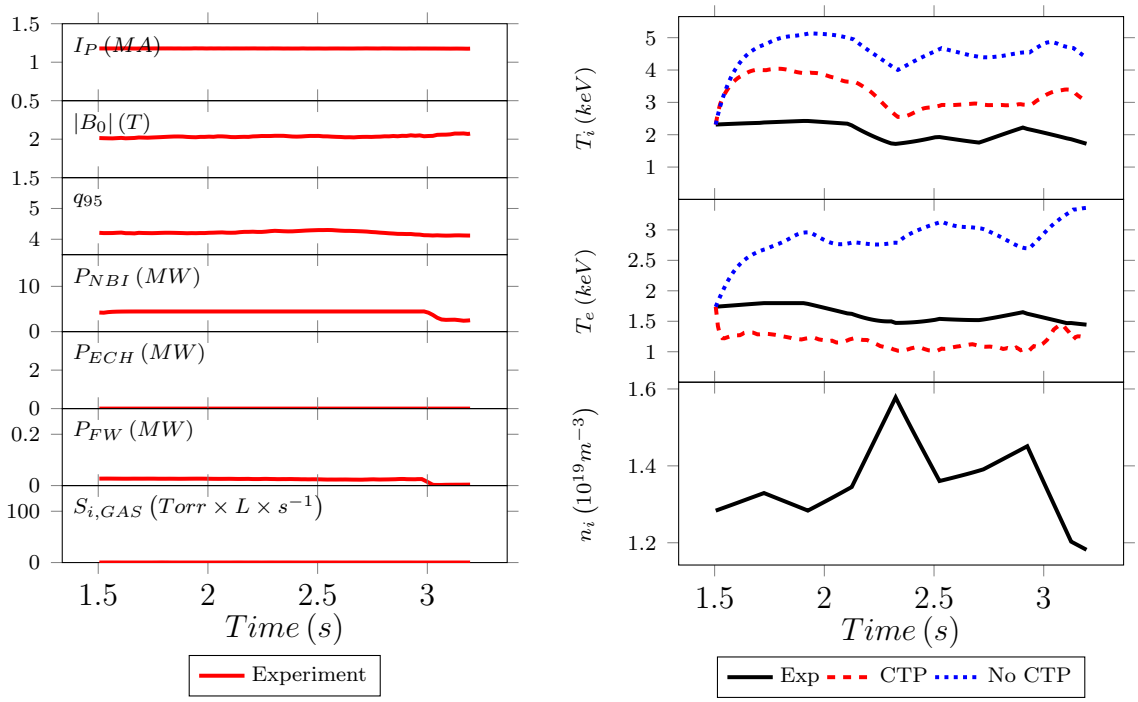


Figure 14: Comparison of simulated and experimental temperatures for DIII-D validation shot 140535, which included widely varying NBI heating power, gradual increases in gas puffing, as well as changes in q_{95} and B_0.

Temperature simulation results for shot 140420 are shown in Figure 15. Generally good agreement was observed for both ion and electron temperatures throughout the shot as ECH power was turned on and gradually reduced. The increase in density observed in this shot despite ECH input was discussed in the previous section.

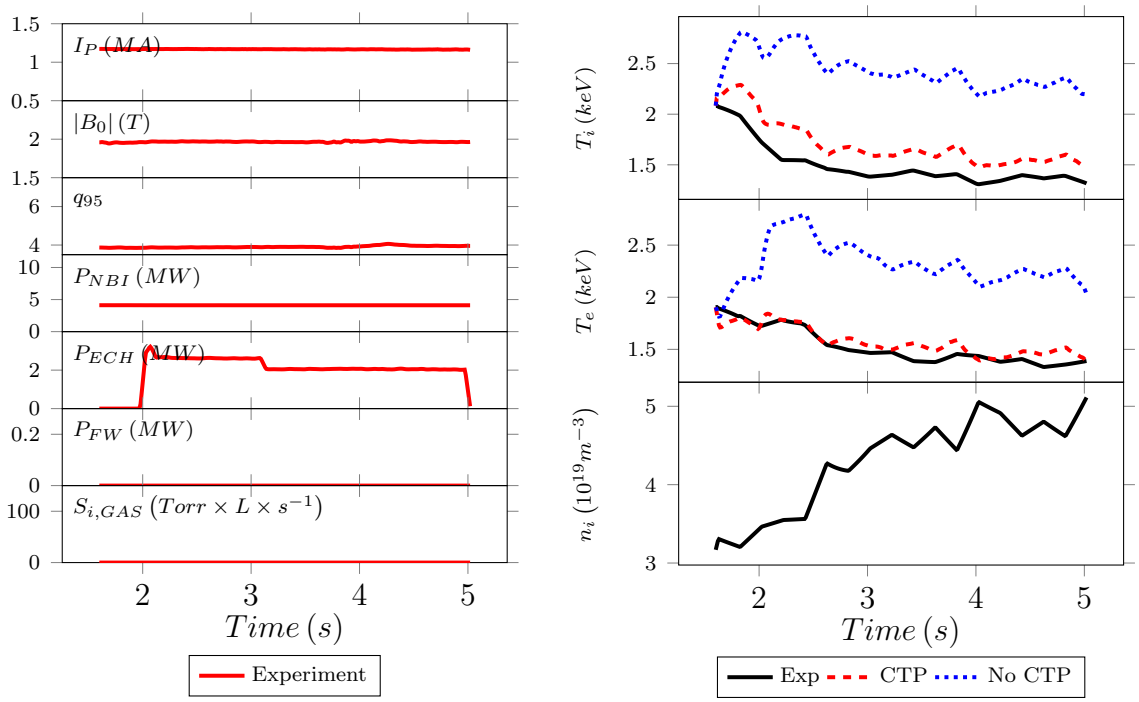


Figure 15: Comparison of simulated and experimental temperatures for DIII-D validation shot 140420.

The final validation shot, shown in Figure 16, is a high collisionality, moderately shaped discharge with significant gas puffing. The simulation showed reasonable quantiative and strong qualitative agreement with experiment. It was observed in many shots, including this one, that the agreement of temperature simulations with experiment tended to degrade with the onset of gas puffing, despite the inclusion of gas puffing magnitude in the tuning parameter regression model. It is, perhaps, not surprising that the relationship between gas puffing and global energy confinement is more complex than is adequately represented in this model. This relationship will be the subject of future research.

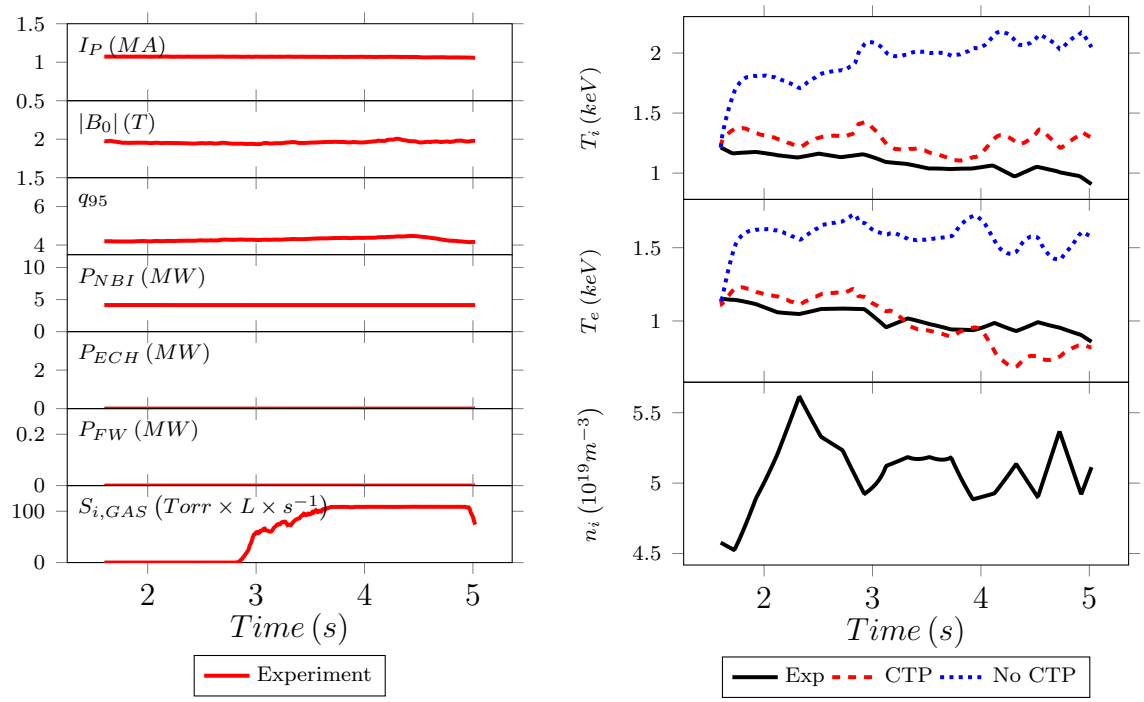


Figure 16: Comparison of simulated and experimental temperatures for DIII-D shot with strong gas puffing (140427).

4.3 Challenges in Density Simulations

As discussed previously, uncertainty in the particle source rate from recycled neutrals made it difficult to accurately simulate changes in ion density. Although simulations of many shots, such as shot 140424 shown in Figure 17 showed strong quantitative agreement with experiment, density simulations clearly represent an opportunity for improvement in the model. One shot that highlights the difficulty in simulating densities is shot 140420, which was described previously. Comparisons of simulated and experimental densities are shown in Figure 18. The simulation model predicted a decrease in density consistent with ECH pumpout; however, the experimental density continued to increase. It is hypothesized that this was likely the result of interactions between the plasma and the lower divertor shelf resulting from the relatively low x-point height in this low triangularity shot. Future improvements to the density simulation model will attempt to include effects like this.

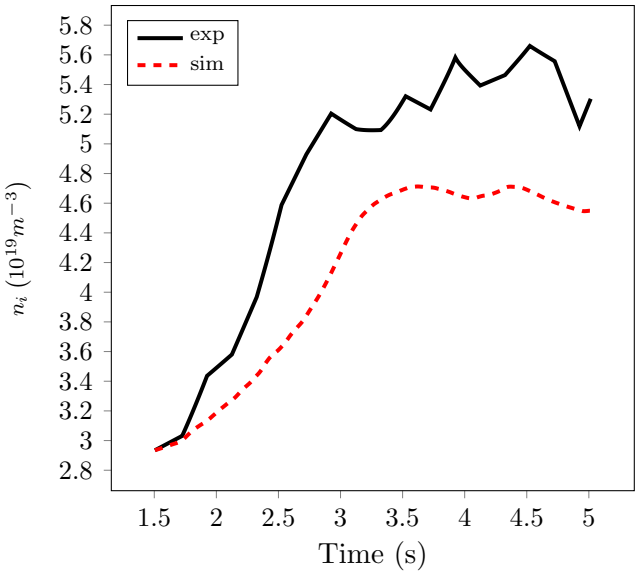


Figure 17: Density simulation for DIII-D shot 140424.

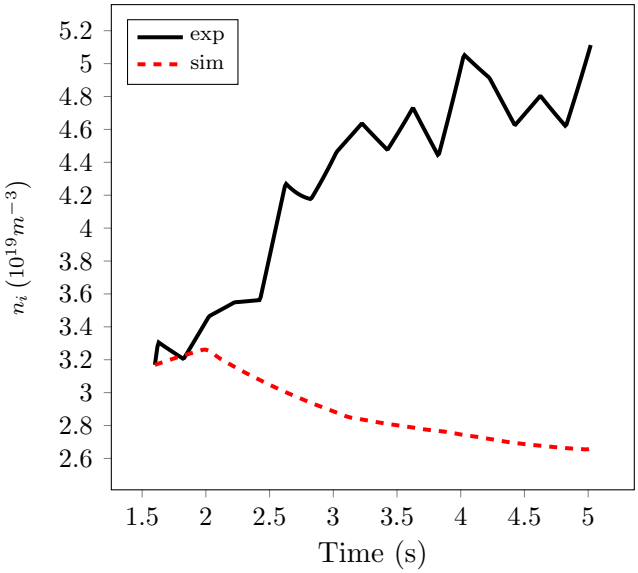


Figure 18: Density simulation for DIII-D shot 140420.

4.4 Calculating a "Goodness of Fit" Parameter

A goodness of fit parameter was developed to describe the overall quality of the simulated densities and temperatures. Each shot was divided into 200 ms time slices, and experimental and simulated quantities were each averaged over each time slice to prevent oscillations about the experimental results from overstating the magnitude

of discrepancies between the simulated and experimental results. The absolute value of the percent relative error in each window was then averaged to obtain an overall goodness of fit parameter for each simulated quantity (G_{Ti}, G_{Te}, and G_{ni}) on each shot. These parameters are reported in Table 6. The analogous average goodness of fit parameters for the simulations in which the confinement tuning parameters were not used is also listed for comparison.

Table 6: Goodness of Fit (G) Parameters Using Simulated and Experimental Densities

	Simulated n_i			Experimental n_i	
Shot	G_{n_i}	G_{T_i}	G_{T_e}	G_{T_i}	G_{T_e}
131190*	10.7%	9.6%	9.5%	14.6%	10.5%
131191	15.1%	3.5%	10.8%	7.7%	6.7%
131195	13.9%	3.1%	10.6%	6.6%	6.4%
131196	22.8%	8.6%	12.3%	4.8%	6.1%
134350	26.3%	14.8%	12.7%	6.3%	5.7%
135837	32.2%	22.4%	23.3%	20.9%	13.2%
135843	30.1%	27.4%	32.5%	28.5%	22.7%
140417	15.2%	5.9%	10.2%	8.8%	5.2%
140418*	17.7%	21.5%	6.4%	8.9%	4.0%
140419	18.9%	17.8%	6.5%	4.6%	4.0%
140420*	35.2%	43.5%	34.3%	14.0%	4.9%
140421	26.0%	26.7%	16.9%	9.0%	4.6%
140422	9.7%	9.8%	4.2%	4.6%	3.9%
140423	13.1%	6.5%	7.3%	9.0%	4.7%
140424	17.5%	28.8%	15.8%	18.0%	10.2%
140425	17.2%	26.6%	14.4%	16.2%	7.6%
140427*	23.5%	17.4%	19.1%	16.4%	9.6%
140428	36.4%	36.6%	30.9%	3.9%	3.7%
140429	32.3%	31.7%	24.3%	5.0%	2.7%
140430	11.9%	11.5%	6.3%	8.2%	1.7%
140431	22.0%	7.9%	8.0%	9.9%	3.5%
140432	22.4%	12.4%	13.8%	3.1%	5.9%
140440	20.1%	33.7%	20.8%	24.4%	17.6%
140535*	29.5%	21.3%	22.9%	28.8%	15.1%
140673	19.0%	16.8%	13.9%	5.3%	9.1%
Mean	**21.5**%	**18.6**%	**15.5**%	**11.5**%	**7.6**
Mean**	**21.5**%	**66.4**%	**45.2**%	**65.1**%	**48.9**

* Validation shots that were not used to construct regression models

** Average of simulations with no tuning parameter, i.e. $\tau_E^{i,e} = \tau_E^{98}$

CHAPTER V

CONCLUSION

We have presented a relatively simple non-linear 0-D plasma dynamics model to represent the effects of changes in various operating and machine parameters on plasma dynamics. The confinement model was tuned to DIII-D using confinement tuning parameters constructed via multiple regression against DIII-D operating and machine parameters and other imputs. Temperature simulations calculated using this model show reasonably good quantitative agreement with a variety of DIII-D discharges and, therefore, give a degree of confidence that this confinement tuning methodology can be used to develop a model for plasma dynamics for ITER and other tokamaks. Fast-wave input power, the value of q_0, plasma elongation, the toroidal magnetic field strength, and several other parameters were shown to have significant effect in the confinement tuning parameter regression models beyond what would be predicted based solely on the ITER-98(y,2) scaling law.

In addition to improving various aspects of the physics model, future work will include extending the model to represent passive and active negative feedback mechanisms that can prevent and limit power excursions in ITER. We plan to enhance the ability of the model to anticipate changes in confinement resulting from MHD activity, ion-orbit loss, thermal instabilities, transitions between confinement modes, and other relevant phenomena, as well as dynamically modeling changes in impurity density resulting from sputtering and improving the neutral recycling model.

APPENDIX A

ADDITIONAL ASPECTS OF PHYSICS MODEL FOR FUTURE USE

A.1 Impurity Sputtering

Sputtering yields are set to zero. However, a sputtering model has been built and included in the model for use in future studies. The sputtering yield is calculated[1] as shown in Equation 25.

$$Y(E) = Q(m_i, m_w, U_B)\, s_n(E)\, g\left(\frac{E_{th}}{E}\right) \tag{25}$$

The nuclear stopping cross section, s_n, is given by

$$s_n(\epsilon) = \frac{3.441\sqrt{\epsilon}\ln\epsilon + 2.718}{1 + 6.355\sqrt{\epsilon} + \epsilon\,(6.882\sqrt{\epsilon} - 1.708)} \tag{26}$$

with $\epsilon = E/E_{TF}$, where E_{TF} is the Thomas-Fermi energy. Threshold effects are accounted for with

$$g = \frac{3.441\sqrt{\epsilon}\ln\epsilon + 2.718}{1 + 6.355\sqrt{\epsilon} + \epsilon\,(6.882\sqrt{\epsilon} - 1.708)} \tag{27}$$

with $E_{th} = U_B/\xi$, where U_B is the binding energy of the wall atom and ξ is the maximum fraction of the energy of the incident atom that can be transferred to a stationary wall atom as shown in Equation 28.

$$\xi = \frac{4m_i m_w}{(m_i + m_w)^2} \tag{28}$$

Q is an empirical yield factor for each material.

A.2 Fusion Reactivity

The plasma reactivity, $\langle \sigma v \rangle_f$, is calculated according to the model developed by Bosch and Hale,[35] which is described in Equations 29 through 31 and uses parameters listed in Table 7.

$$\langle \sigma v \rangle_f = C_1 \sqrt{\frac{\xi}{m_r c^2 T^3}} e^{-3\xi} \tag{29}$$

$$\theta = \frac{T}{1 - \frac{T(C_2 + T(C_4 + TC_6))}{1 + T(C_3 + T(C_5 + TC_7))}} \tag{30}$$

$$\xi = \left(\frac{B_G^2}{4\theta} \right) \tag{31}$$

Table 7: Parameters for Fusion Reactivity Calculation

$Coefficient$	$T\,(d,n)^4\,He$	$D\,(d,p)\,T$	$D\,(d,n)^3\,He$
$B_G\left(\sqrt{keV}\right)$	34.3827	31.3970	31.3970
$m_r c^2$	1.124656×10^6	9.37814×10^5	9.37814×10^5
C_1	1.17302×10^{-9}	5.65718×10^{-12}	5.43360×10^{-12}
C_2	1.51361×10^{-2}	3.41267×10^{-3}	5.85778×10^{-3}
C_3	7.51886×10^{-2}	1.99167×10^{-3}	7.68222×10^{-3}
C_4	4.60643×10^{-3}	0.0	0.0
C_5	1.35000×10^{-2}	1.05060×10^{-5}	-2.96400×10^{-6}
C_6	-1.0650×10^{-4}	0.0	0.0
C_7	1.36600×10^{-5}	0.0	0.0

APPENDIX B

ADDITIONAL SHOT SIMULATIONS

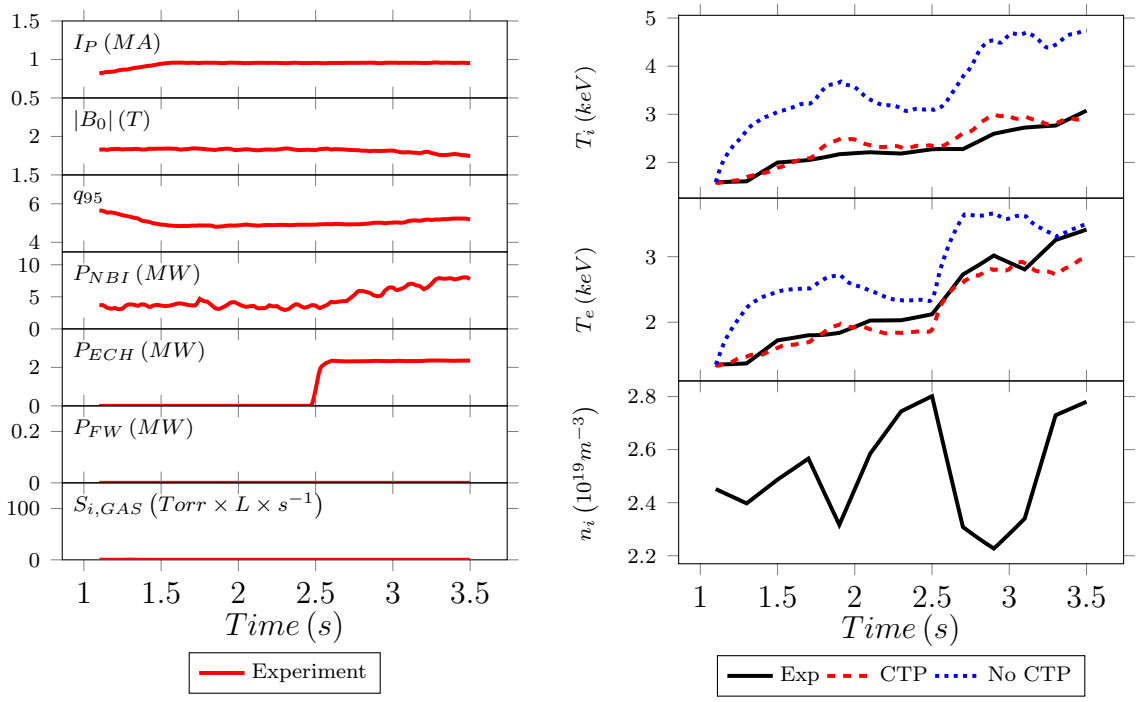


Figure 19: Comparison of simulated and experimental temperatures for DIII-D shot 131191. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

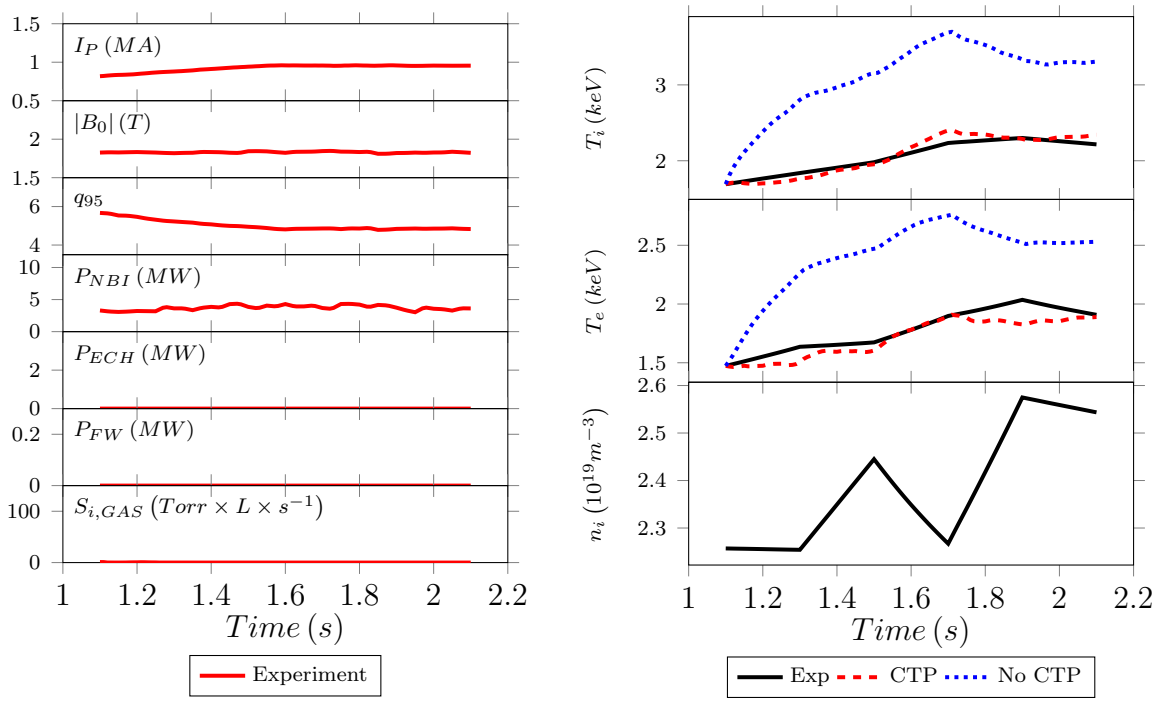


Figure 20: Comparison of simulated and experimental temperatures for DIII-D shot 131195. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

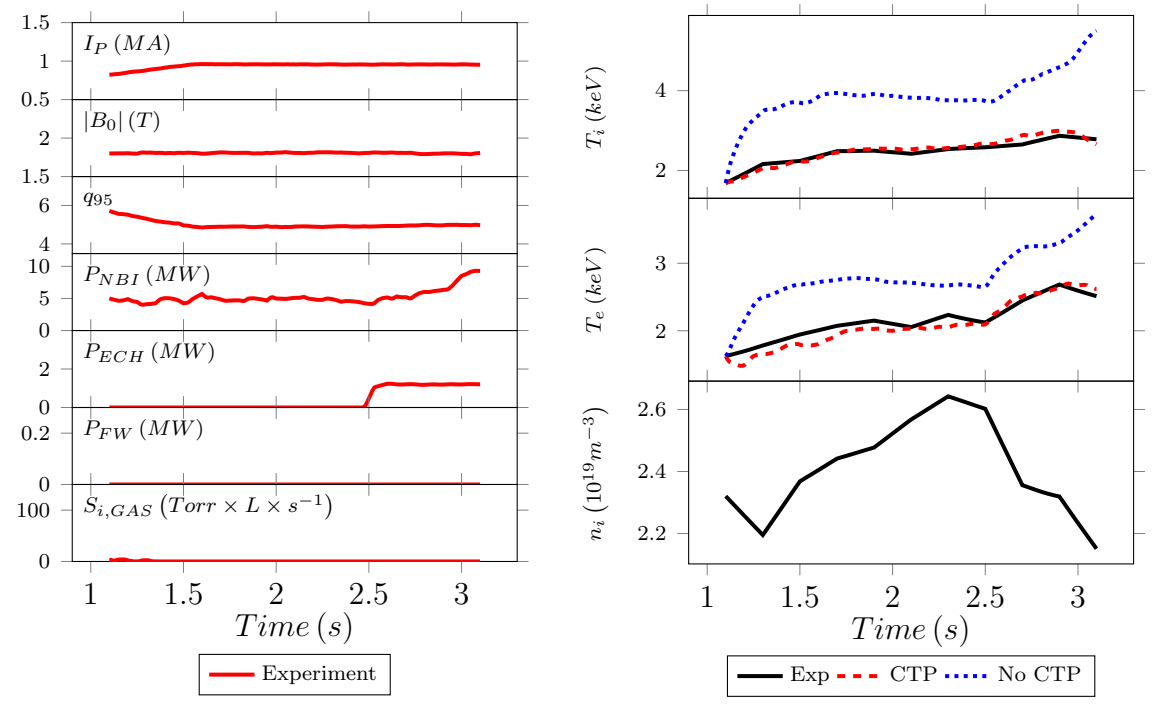


Figure 21: Comparison of simulated and experimental temperatures for DIII-D shot 131196. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

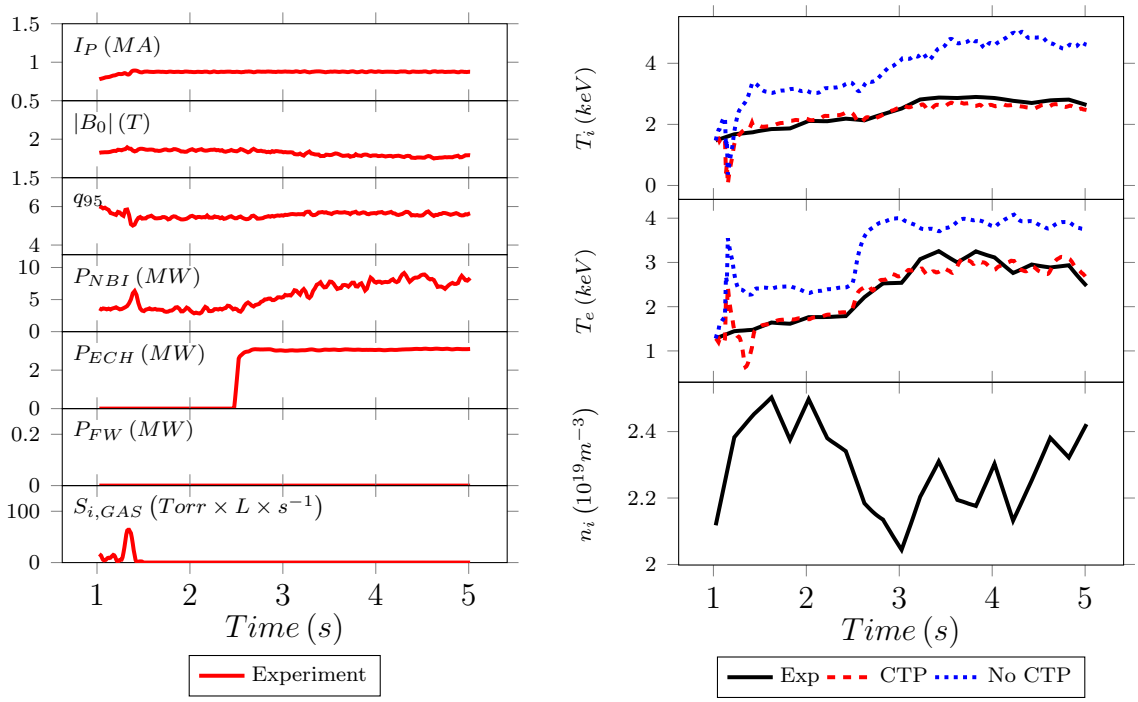


Figure 22: Comparison of simulated and experimental temperatures for DIII-D shot 134350. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

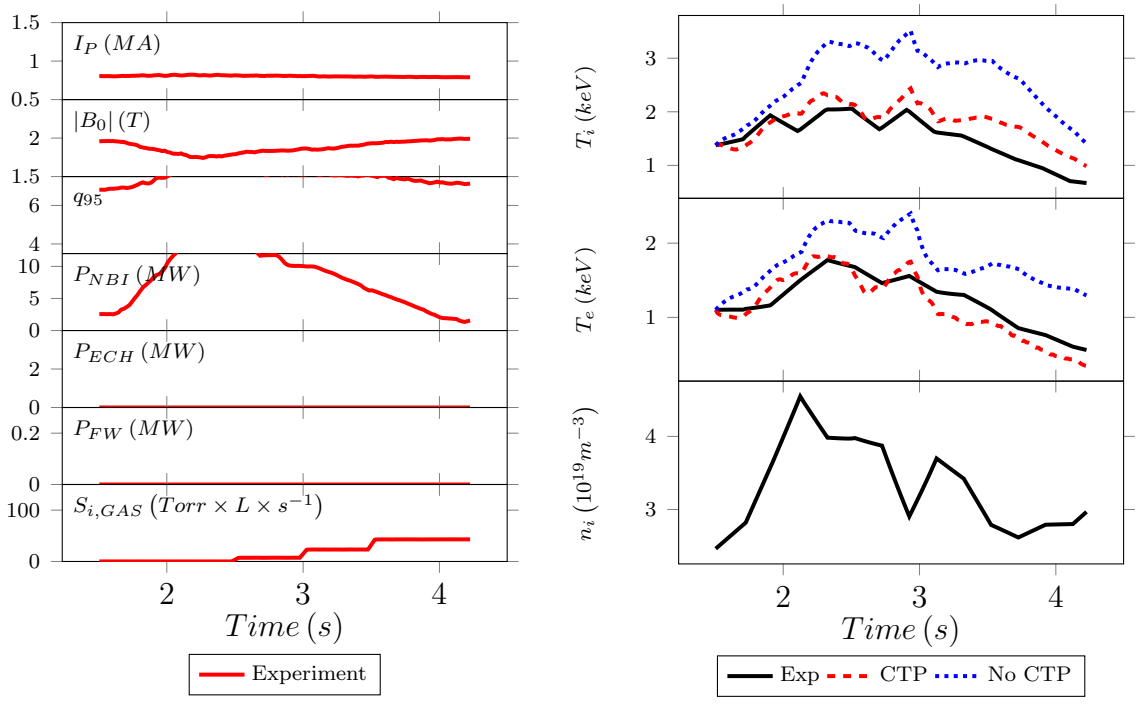


Figure 23: Comparison of simulated and experimental temperatures for DIII-D shot 135837. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

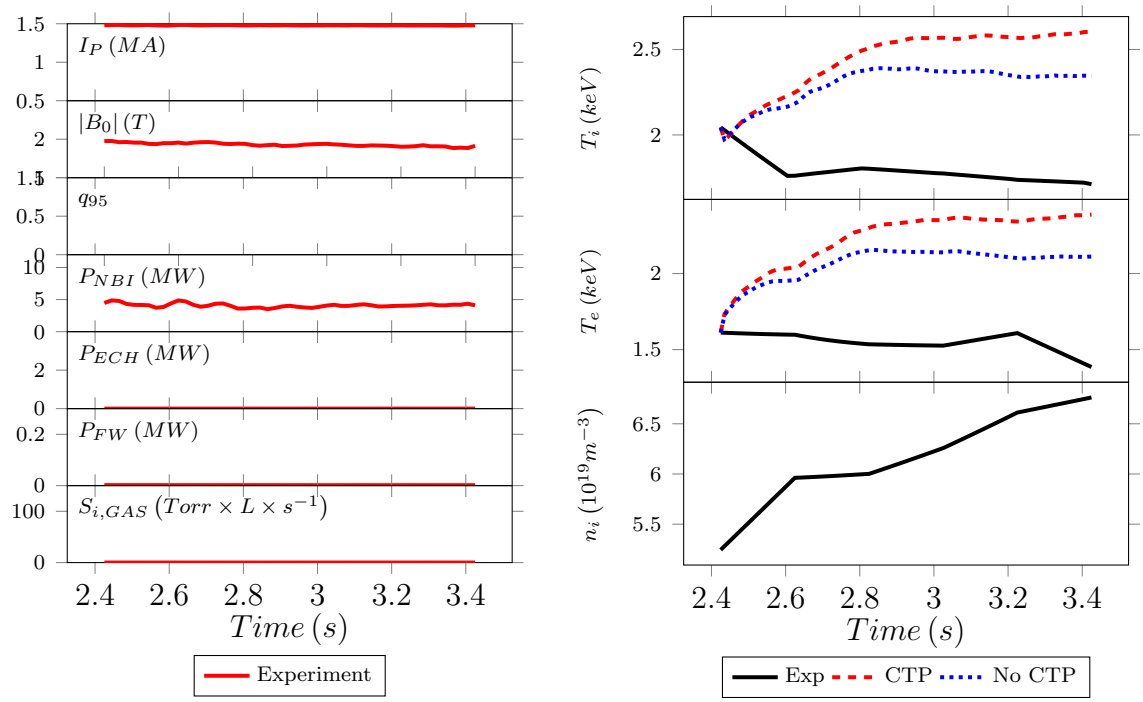


Figure 24: Comparison of simulated and experimental temperatures for DIII-D shot 135843. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

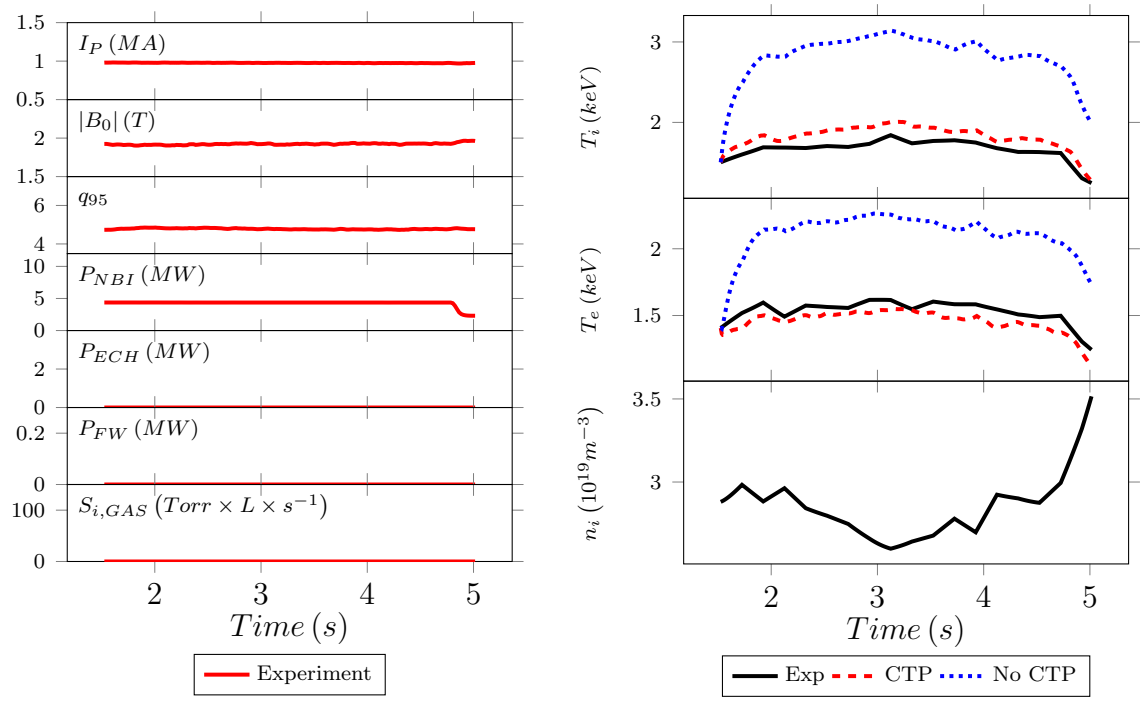


Figure 25: Comparison of simulated and experimental temperatures for DIII-D shot 140417. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

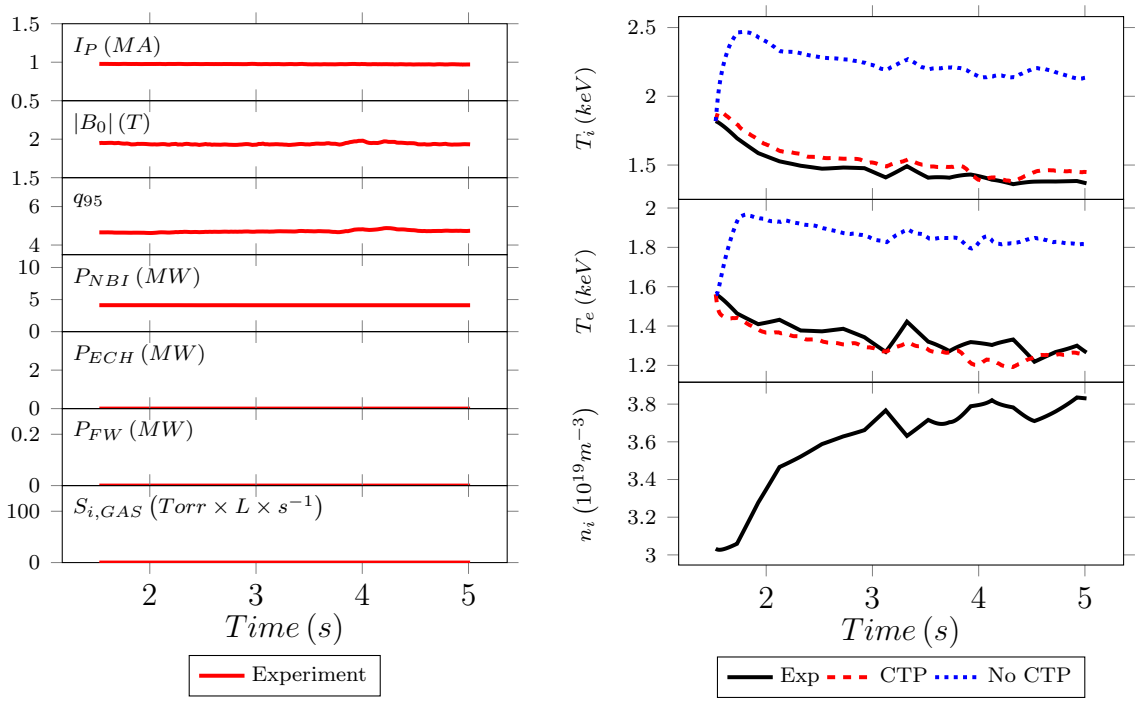


Figure 26: Comparison of simulated and experimental temperatures for DIII-D shot 140419. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

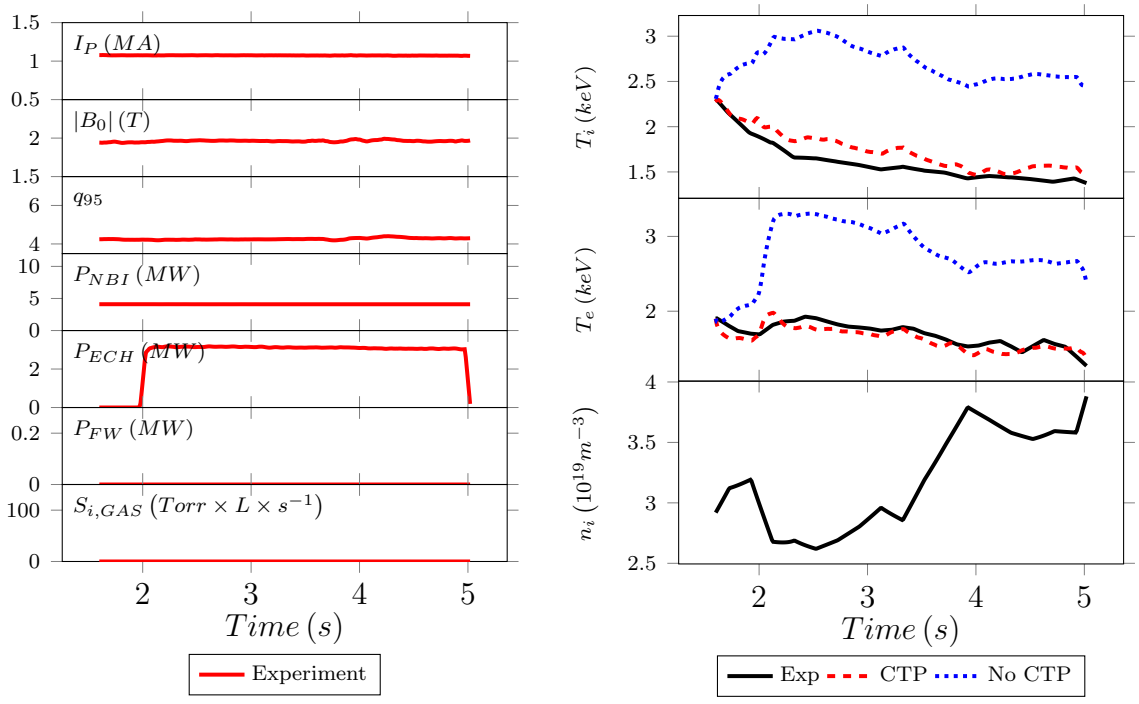


Figure 27: Comparison of simulated and experimental temperatures for DIII-D shot 140421. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

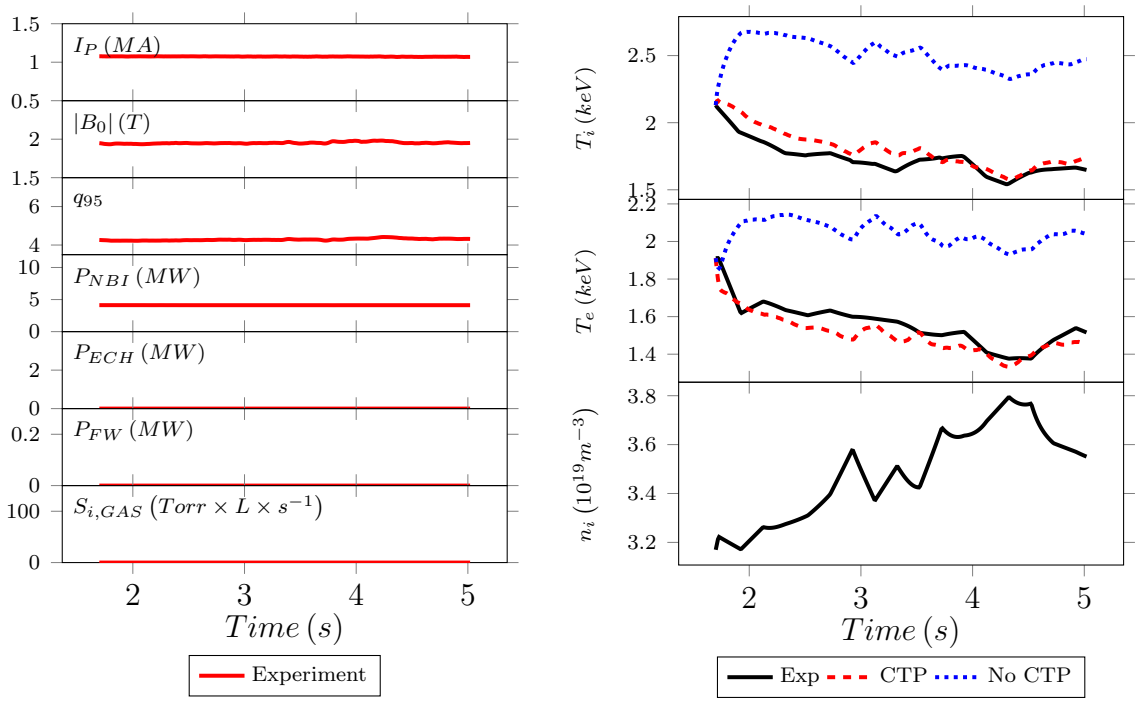


Figure 28: Comparison of simulated and experimental temperatures for DIII-D shot 140422. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

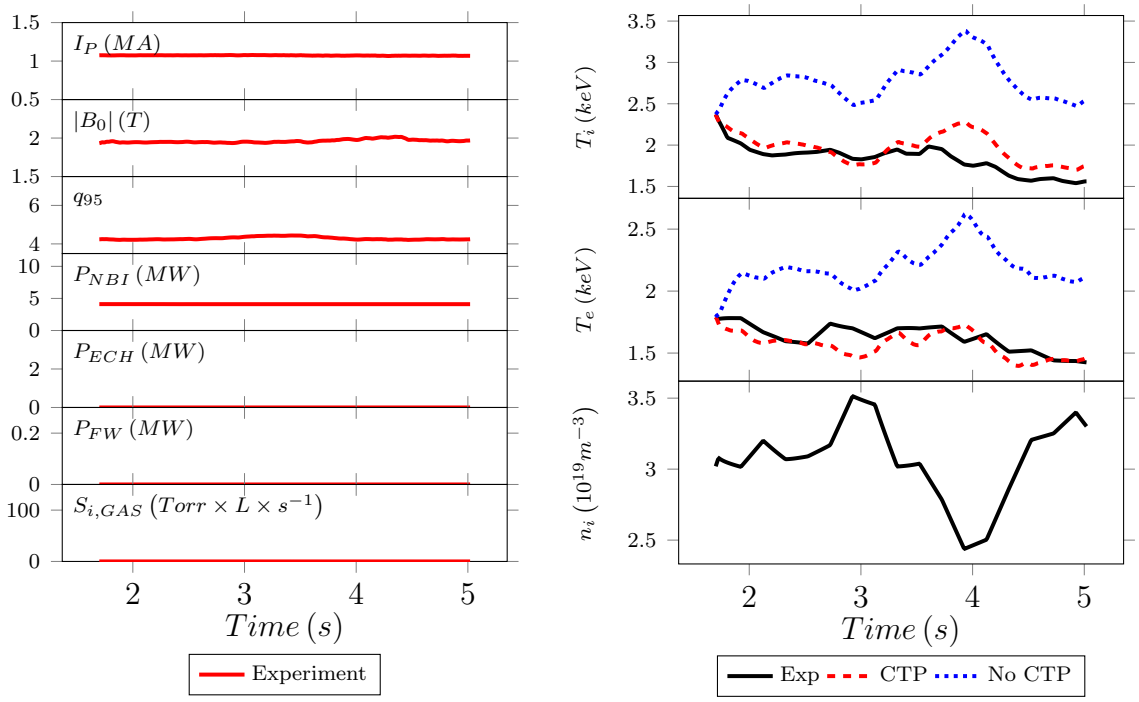


Figure 29: Comparison of simulated and experimental temperatures for DIII-D shot 140423. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

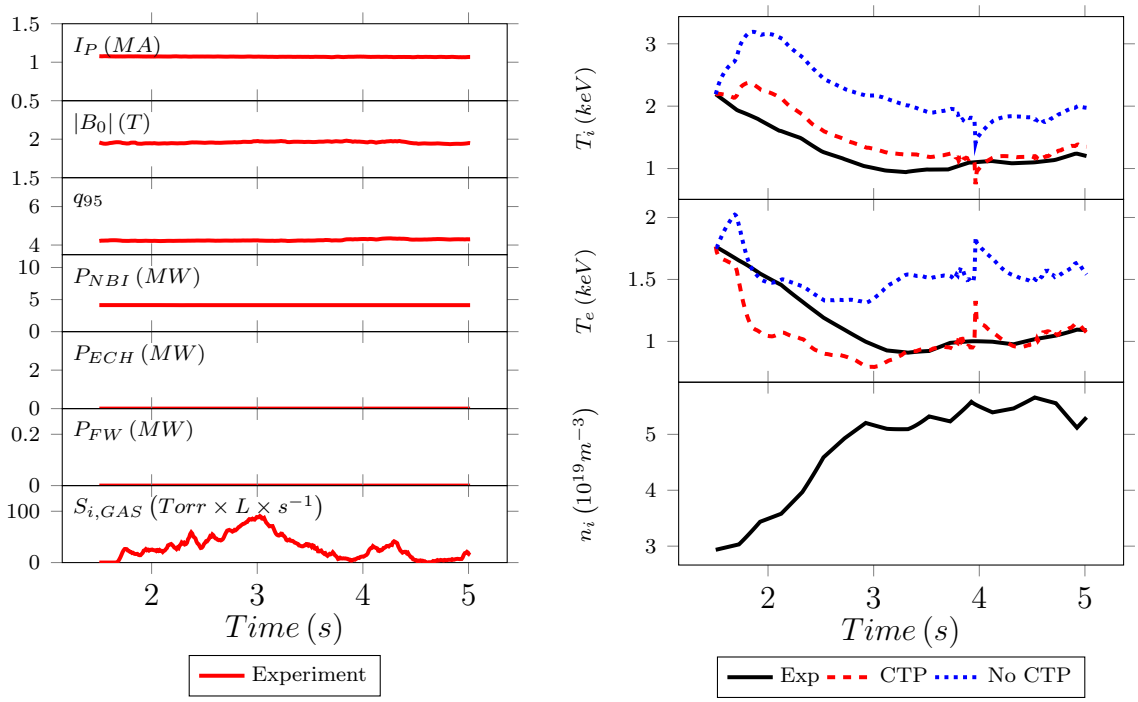


Figure 30: Comparison of simulated and experimental temperatures for DIII-D shot 140424. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

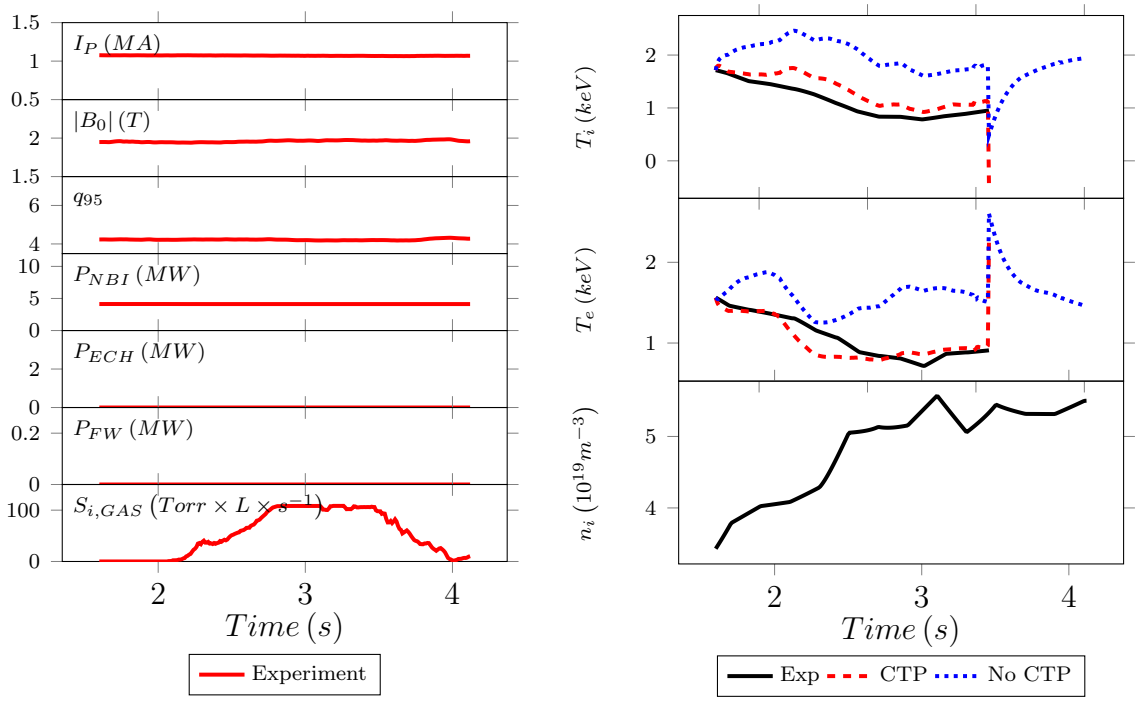


Figure 31: Comparison of simulated and experimental temperatures for DIII-D shot 140425. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

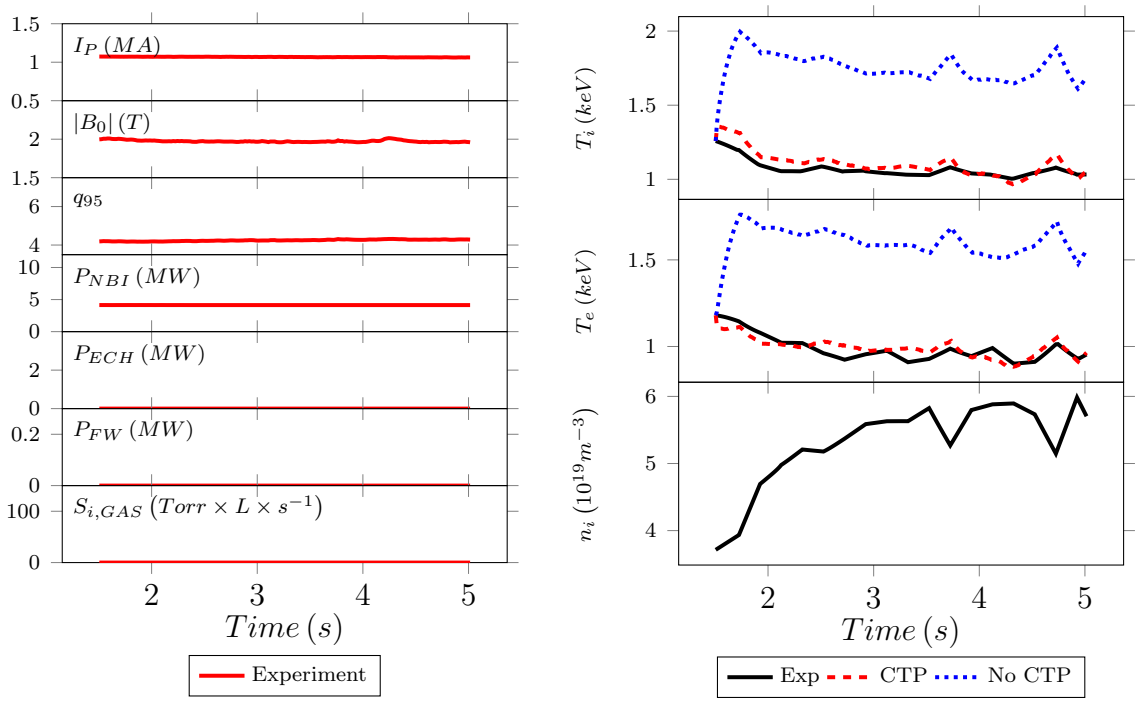


Figure 32: Comparison of simulated and experimental temperatures for DIII-D shot 140428. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

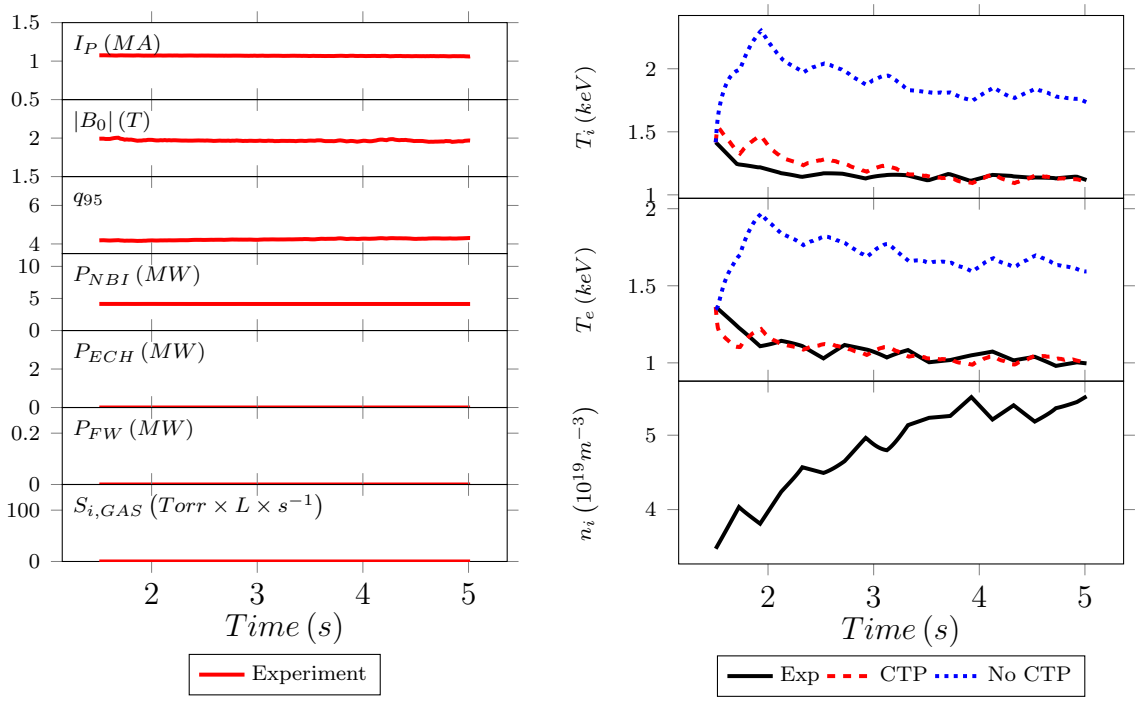


Figure 33: Comparison of simulated and experimental temperatures for DIII-D shot 140429. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

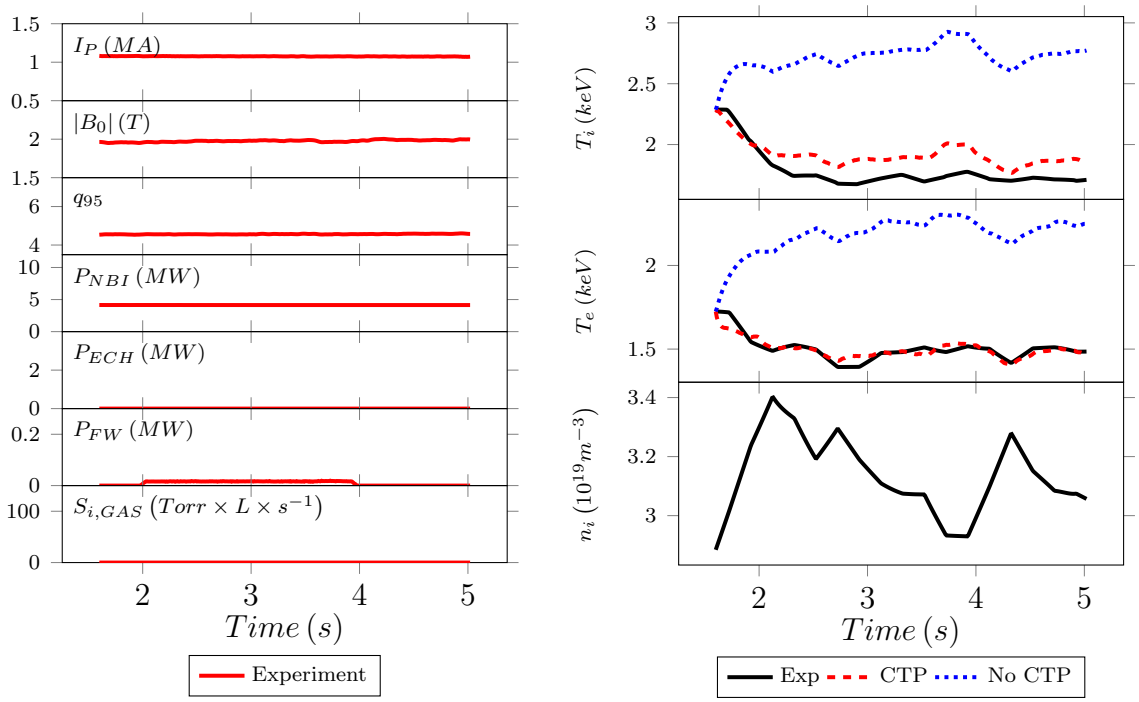


Figure 34: Comparison of simulated and experimental temperatures for DIII-D shot 140430. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

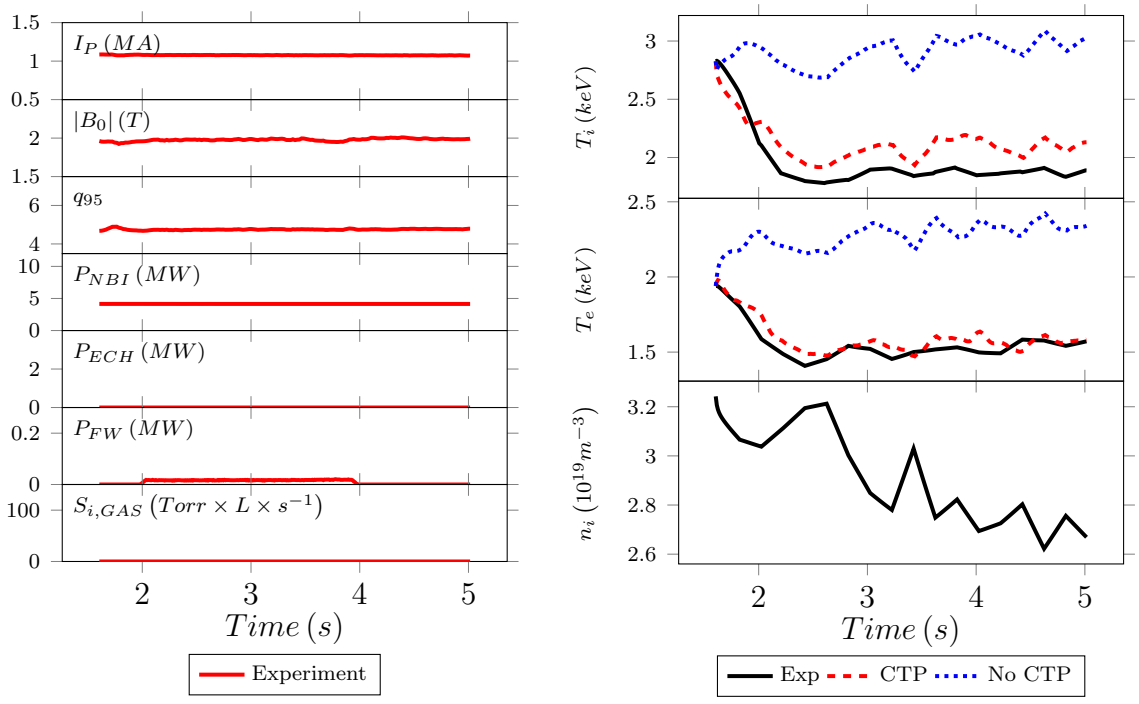


Figure 35: Comparison of simulated and experimental temperatures for DIII-D shot 140431. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

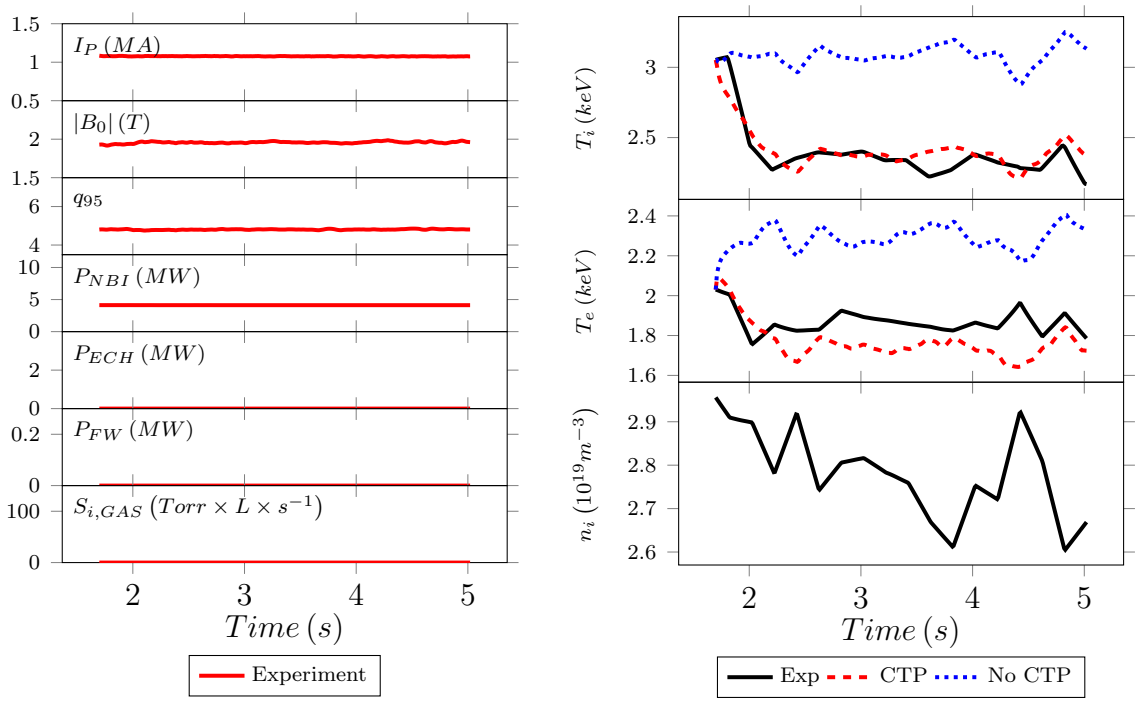


Figure 36: Comparison of simulated and experimental temperatures for DIII-D shot 140432. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

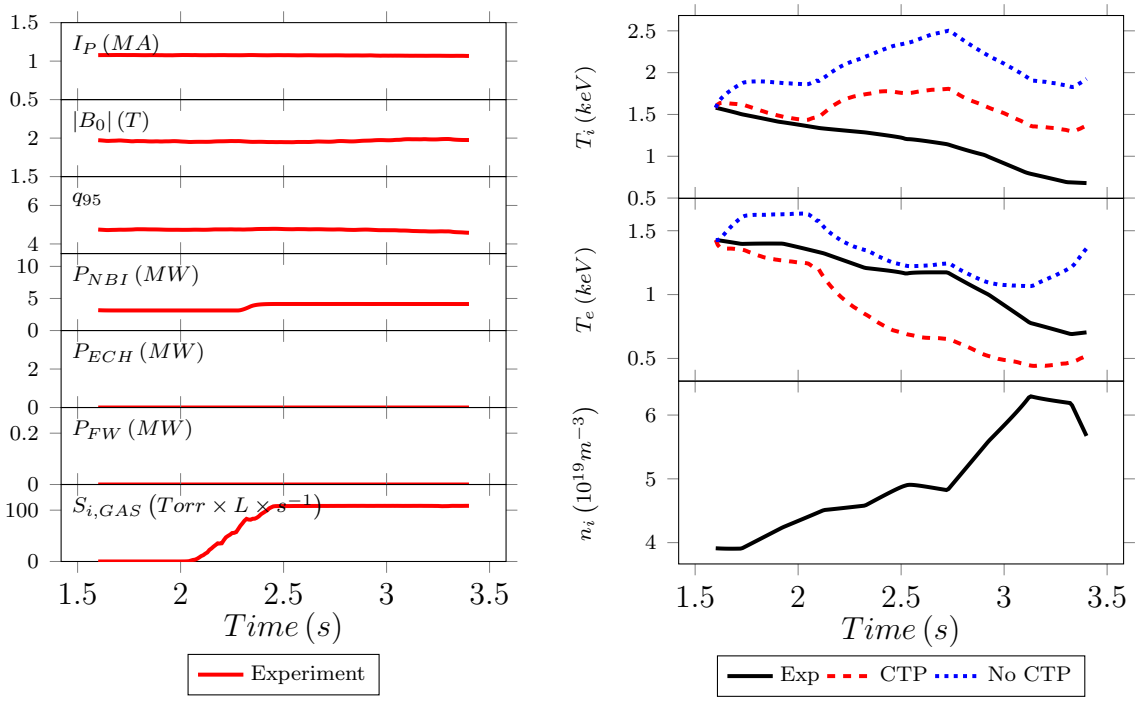


Figure 37: Comparison of simulated and experimental temperatures for DIII-D shot 140440. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

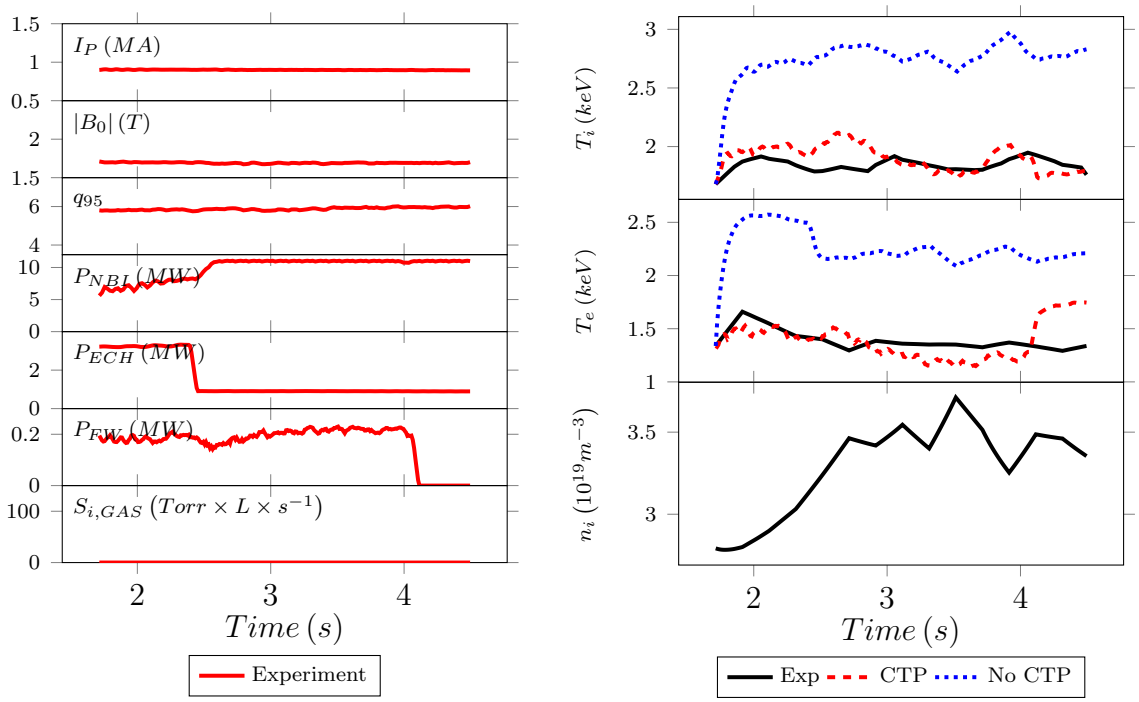


Figure 38: Comparison of simulated and experimental temperatures for DIII-D shot 140673. Background plasma information is shown on the left. Density was set to experiment and is shown for reference.

APPENDIX C

GTBURN SOURCE CODE

```
GT_BURN2: variables.o allocate_arrays.o input.o data_smooth.o timeseries.o initial_n_t.o fast_ion.o radial.o simulation.o f2e_prep_timeseries.o GT_BURN2.o
	gfortran -o GT_BURN2 variables.o allocate_arrays.o input.o data_smooth.o timeseries.o initial_n_t.o fast_ion.o radial.o simulation.o f2e_prep_timeseries.o GT_BURN2.o -llapack
variables.mod: variables.o variables.f95
	gfortran -c -ffree-line-length-0 variables.f95 -llapack
variables.o: variables.f95
	gfortran -c -ffree-line-length-0 variables.f95 -llapack
allocate_arrays.o: variables.mod allocate_arrays.f95
	gfortran -c -ffree-line-length-0 allocate_arrays.f95 -llapack
input.o: variables.mod input.f95
	gfortran -c -ffree-line-length-0 input.f95 -llapack
data_smooth.o: variables.mod data_smooth.f95
	gfortran -c -ffree-line-length-0 data_smooth.f95 -llapack
timeseries.o: variables.mod timeseries.f95
	gfortran -c -ffree-line-length-0 timeseries.f95 -llapack
initial_n_t.o: variables.mod initial_n_t.f95
	gfortran -c -ffree-line-length-0 initial_n_t.f95 -llapack
fast_ion.o: variables.mod fast_ion.f95
	gfortran -c -ffree-line-length-0 fast_ion.f95 -llapack
radial.o: variables.mod radial.f95
	gfortran -c -ffree-line-length-0 radial.f95 -llapack
simulation.o: variables.mod simulation.f95
	gfortran -c -ffree-line-length-0 simulation.f95 -llapack
f2e_prep_timeseries.o: variables.mod f2e_prep_timeseries.f95
	gfortran -c -ffree-line-length-0 f2e_prep_timeseries.f95 -llapack
GT_BURN2.o: variables.mod GT_BURN2.f95
	gfortran -c -ffree-line-length-0 GT_BURN2.f95 -llapack
clean:
	rm *.o *.mod
# End of the makefile
```

```
Program GT_BURN2
Use Variables
implicit NONE
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Input/output log
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       Unit    File
!
!       1       Global Input File
!       2       Shot Input File
!       3       Global Output File
!       4       Global Output File 2
!
!       5       RESERVED FOR I/O TO USER
!       6       RESERVED FOR I/O TO USER
!
!       7       Raw data file from gadat2 and IDL script
!       8       rad outfile, only done during initial_n_t
!       9       outfile4, show results of mode 1 simulation compared with "experiment" from mode 4
!       10      outfile8, used to transfer data from mode 4 to mode 1
!       11      outfile2, regular shot output
!       13      raw synced output, outfile1
!       14      Raw output
!       15      Fast Ion output
!       16      Event change output, outfile_global3
!       17      Goodness of fit output!
!       18      outfile9 - Output radial timeseries data (compare av quantities with profile)
!       19      global statistics (max, min, etc.)
!       20      smoothed version of the raw data
!       21      global output of ranges of various parameters (ECH, NBI, etc.) One line per shot.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

namelist /input_global/ delta_t, shots, shotnum, S_rec_factor, S_reem_factor, output_mode,
H_find_mode, Sim_mode, Q_ie_mult, gas_av, seg_width, del_source, raw_output, raw_synced_output,
h_reuse, S_i_rec_fit, H_i_part_1, Hei1, Hei2, Hei3, Hei4, Hei5, Hei6, Hei7, Hei8, Hei9, Hei10,
Hei11, Hei12, Hei13, Hei14, Hei15, Hei16, Hei17, Hei18, Hei19, Hei20, Hee1, Hee2, Hee3, Hee4, Hee5,
Hee6, Hee7, Hee8, Hee9, Hee10, Hee11, Hee12, Hee13, Hee14, Hee15, Hee16, Hee17, Hee18, Hee19,
Hee20, Hpi1, Hpi2, Hpi3, Hpi4, Hpi5, Hpi6, Hpi7, Hpi8, Hpi9, Hpi10, Hpi11, Hpi12, Hpi13, Hpi14,
Hpi15, Hpi16, Hpi17, Hpi18, Hpi19, Hpi20

namelist /input_shot/ shot, start_time, end_time, up_divertor, event_mode, event_time, ni_sim,
Ti_sim, Te_sim, z, fz, Hz, RMP_start, RMP_end, cleanup_start1, cleanup_end1, skip_lt1, skip_gt1,
cleanup_start2, cleanup_end2, skip_lt2, skip_gt2, cleanup_start3, cleanup_end3, skip_lt3, skip_gt3,
cleanup_start4, cleanup_end4, skip_lt4, skip_gt4, cleanup_start5, cleanup_end5, skip_lt5, skip_gt5,
cleanup_start6, cleanup_end6, skip_lt6, skip_gt6, cleanup_start7, cleanup_end7, skip_lt7, skip_gt7,
cleanup_start8, cleanup_end8, skip_lt8, skip_gt8, cleanup_start9, cleanup_end9, skip_lt9, skip_gt9,
cleanup_start10, cleanup_end10, skip_lt10, skip_gt10, cleanup_start11, cleanup_end11, skip_lt11,
skip_gt11, cleanup_start12, cleanup_end12, skip_lt12, skip_gt12, cleanup_start13, cleanup_end13,
skip_lt13, skip_gt13, cleanup_start14, cleanup_end14, skip_lt14, skip_gt14, cleanup_start15,
cleanup_end15, skip_lt15, skip_gt15, cleanup_start16, cleanup_end16, skip_lt16, skip_gt16,
cleanup_start17, cleanup_end17, skip_lt17, skip_gt17, cleanup_start18, cleanup_end18, skip_lt18,
skip_gt18, cleanup_start19, cleanup_end19, skip_lt19, skip_gt19, cleanup_start20, cleanup_end20,
skip_lt20, skip_gt20, cleanup_start21, cleanup_end21, skip_lt21, skip_gt21, cleanup_start22,
cleanup_end22, skip_lt22, skip_gt22, cleanup_start23, cleanup_end23, skip_lt23, skip_gt23,
cleanup_start24, cleanup_end24, skip_lt24, skip_gt24, cleanup_start25, cleanup_end25, skip_lt25,
skip_gt25, cleanup_start26, cleanup_end26, skip_lt26, skip_gt26, cleanup_start27, cleanup_end27,
skip_lt27, skip_gt27, cleanup_start28, cleanup_end28, skip_lt28, skip_gt28, cleanup_start29,
cleanup_end29, skip_lt29, skip_gt29, cleanup_start30, cleanup_end30, skip_lt30, skip_gt30,
ne_seg_mode, ne_time, ne_0_seg, ne_5_seg, ne_9_seg, Ti_seg_mode, Ti_time, Ti_0_seg, Ti_5_seg,
Ti_9_seg, Te_seg_mode, Te_time, Te_0_seg, Te_5_seg, Te_9_seg
!START PROGRAM
Print *,'starting GT_BURN2'

!ALLOCATE AND ZEROIZE ALL ARRAYS
Print *,'Allocating arrays'
Call Allocate_arrays

!GET GLOBAL INPUT FILE INFO
Print *,'Getting global input file'
!call getcwd(cwd)
```

```
infile_global = '/home/max/SO_Max/PhD/GT_BURN2/input/INPUT_GLOBAL.txt'
Open (unit = 1,FILE = INFILE_GLOBAL)
Read (1,nml=input_global)
Close (1)

Do i=1,shots
!       write (shotnum_string(i),'(I6)') shotnum(i)
        shotnum_string(i) = shotnum(i)
End Do

!START MAIN PROGRAM DO LOOP TO LOOP THROUGH SHOTS
Do filecount=1,shots
        Print *,''
        Print *,'working on shot ',shotnum(filecount),'  ',filecount,' of ',shots,' shots'
        Print *,'sim_mode = ',sim_mode

        !ASSIGN NAMES TO INPUT AND OUTPUT FILES FOR THE SHOT
        infile1  = '/home/max/SO_Max/PhD/GT_BURN2/input/INPUT_'//shotnum(filecount)
        Open (unit = 2,FILE = infile1)
        Read (2,nml=input_shot)
        Close (2)

        ni_sim = 1
        Ti_sim = 1
        Te_sim = 1

        !MODIFY START TIME IF NECESSARY
        !If (sim_mode.eq.4) Then
                start_time = start_time + 0.3
                !end_time = end_time - 0.2
        !End If

        If (ni_sim.eq.1.and.Ti_sim.eq.1) Then
                ni_sim_path = "_nt"
        Else if (ni_sim.eq.1.and.Ti_sim.eq.0) Then
                ni_sim_path = "_n_"
        Else if (ni_sim.eq.0.and.Ti_sim.eq.1) Then
                ni_sim_path = "_t_"
        End If

        outfile1 =              '/home/max/SO_Max/PhD/GT_BURN2/output/output_s/outfile1_'//shotnum
(filecount)//'.txt'
        outfile2 =              '/home/max/SO_Max/PhD/GT_BURN2/output/output'//trim(ni_sim_path)//'/
outfile2_'//shotnum(filecount)//'.txt'
        outfile3 =              '/home/max/SO_Max/PhD/GT_BURN2/output/output'//trim(ni_sim_path)//'/
outfile3_'//shotnum(filecount)//'.txt'
        outfile4 =              '/home/max/SO_Max/PhD/GT_BURN2/output/output'//trim(ni_sim_path)//'/
outfile4_'//shotnum(filecount)//'.txt'
        outfile5 =              '/home/max/SO_Max/PhD/GT_BURN2/output/output'//trim(ni_sim_path)//'/
outfile5_'//shotnum(filecount)//'.txt'
        outfile6 =              '/home/max/SO_Max/PhD/GT_BURN2/output/output'//trim(ni_sim_path)//'/
outfile6_'//shotnum(filecount)//'.txt'
        outfile7 =              '/home/max/SO_Max/PhD/GT_BURN2/output/output'//trim(ni_sim_path)//'/
outfile7_'//shotnum(filecount)//'.txt'
        outfile8 =              '/home/max/SO_Max/PhD/GT_BURN2/output/output_s/outfile8_'//shotnum
(filecount)//'.txt'
        outfile9 =              '/home/max/SO_Max/PhD/GT_BURN2/output/output_s/outfile9_'//shotnum
(filecount)//'.txt'
        rad_outfile =           '/home/max/SO_Max/PhD/GT_BURN2/output/output'//trim(ni_sim_path)//'/
rad_outfile_'//shotnum(filecount)//'.txt'
        fast_ion_out =          '/home/max/SO_Max/PhD/GT_BURN2/output/output'//trim(ni_sim_path)//'/
fast_ion_'//shotnum(filecount)//'.txt'
        datafile =              '/home/max/SO_Max/PhD/GT_BURN2/input/timeseries/'//trim
(shot)//'_data.txt'
        datafile_smooth =       '/home/max/SO_Max/PhD/GT_BURN2/input/timeseries/'//trim
(shot)//'_data_s.txt'

        !MAKE ANY CHANGES TO THE INPUT FILE HERE
        If (end_time.gt.5500) Then
                end_time = 5.500 !having a problem with maxing out some array if this is much
```

```
longer. Something I need to deal with later.
103         End If
104
105         If (event_mode.eq.1) Then
106                 start_time = event_time - .250
107                 end_time = event_time + .250
108         End If
109
110         i_pre = 0 !10000 for .1 sec
111         exit_flag=0
112
113         !READ IN INPUT TIMESERIES DATA
114         Print *,'starting timeseries'
115         Call timeseries
116
117         !SET INITIAL CONDITIONS IF IN MODE 1. POPULATE TIMESERIES OF Q_ie, nu, densities, and
    temperatures if in mode 4
118         !If (sim_mode.eq.1) Then
119         !       Print *,'starting initial_n_t'
120         !       Call initial_n_t
121         !Else
122                 Print *,'starting f2e_prep_timeseries' !f2e = force to experiment. Used to back-
    calculate implied confinement parameters
123                 Call f2e_prep_timeseries
124         !End If
125
126         Print *,'    starting simulation'
127         Call simulation
128
129         Print *,'    Writing output'
130
131         !SET UP GLOBAL OUTPUT FILES (CONTAIN OUTPUT FROM ALL SHOTS)
132         If (filecount.eq.1) Then
133                 outfile_global =        '/home/max/SO_Max/PhD/GT_BURN2/output/output_g/
    outfile_global.txt'
134                 outfile_global2 =       '/home/max/SO_Max/PhD/GT_BURN2/output/output_g/
    outfile_global2.txt'
135                 outfile_global3 =       '/home/max/SO_Max/PhD/GT_BURN2/output/output_g/
    outfile_global3.txt'
136                 outfile_global4 =       '/home/max/SO_Max/PhD/GT_BURN2/output/output_g/
    outfile_global4.txt'
137                 outfile_global5 =       '/home/max/SO_Max/PhD/GT_BURN2/output/output_g/
    outfile_global5.txt'
138                 outfile_global6 =       '/home/max/SO_Max/PhD/GT_BURN2/output/output_g/
    outfile_global6.txt'
139
140                 Open (unit = 3,FILE = outfile_global)
141
142                 Open (unit = 16,FILE = outfile_global3)
143                 Open (unit = 17,FILE = outfile_global4)
144                 Open (unit = 19,FILE = outfile_global5)
145                 Open (unit = 21,FILE = outfile_global6)
146                 Write (3,*) "shot IP BT0_abs ni_19 ne_19 Ti Te Ti/Te ne/Te^2 ni/Ti^2 sqrt(Ti)/B
    H_i_energy log(H_i_energy) H_e_energy log(H_e_energy) H_i_part tau_e tau_i_energy tau_e_energy
    tau_i_part H98 H98*tau_e Q_ie1 Q_ie2 Q_ie3 H98_calc taue_raw tauth_raw taue_raw/taue tauth_raw/taue
    taueh98y2_Raw densv2(1) densv2_calc"
147                 If (sim_mode.eq.4) Then
148                         Open (unit = 4,FILE = outfile_global2)
149                         Write (4,*) "shot t_exp PNBI/volume nbi_counter_frac nbi_short_frac
    Paux_e_vol Paux_i_vol q95_raw q0_raw IP/xsec_area BT0_abs S_i_gas/volume tri_divert tri_nondivert
    ELONG_A Q_ie ni Ti_kev Te_kev Ti_Te conf_loss H_i_energy H_e_energy H_i_part S_i_rec_vol_tot"
150                 End If
151                 Write (16,*) "PNBI_vol Paux_e_vol Paux_i_vol S_i_gas_vol ni Ti_keV Te_kev Ti_Te
    H_i_energy H_e_energy H_i_part "
152                 Write (17,*) "shot ni Ti Te"
153                 Write (19,*) "shot t_exp PNBI nbi_counter_frac Paux_e Paux_i q95_raw IP BT0_abs
    S_i_gas tri_divert tri_nondivert ELONG_A ni Ti_keV Te_keV"
154                 Write (21,*) "shot start_time end_time ECH_min ECH_max FW_min FW_max GAS_min
    GAS_max NBI_min NBI_max B0_min B0_max q95_min q95_max ni_min ni_max Te_min Te_max Ti_min Ti_max"
155         End If
```

```
156
157         If (event_mode.eq.1) Then
158                 Write (16,"(4(F30.3))",advance='no') (event_change(j), j=1,4)
159                 !Write (16,"(28(F30.3))",advance='no') (event_change_rel(j), j=1,28)
160                 Write (16,"(4(F30.3))",advance='no') (event_pre(j), j=5,8)
161                 Write (16,"(3(F30.3))",advance='no') (event_change(j), j=9,11)
162                 Write (16,*) ''
163         End If
164
165         !Write global output. This is done for all output modes.
166         Write (3,*) shotnum(filecount), IP(1), BT0_abs(1), ni_19(1), ne_19(1), Ti_keV(1), Te_keV
    (1), Ti_keV(1)/Te_keV(1), ne_19(1)/Te_keV(1)**2, ni_19(1)/Ti_keV(1)**2,  sqrt(Ti_keV(1))/BT0_abs
    (1), H_i_energy(1), log(H_i_energy(1)), H_e_energy(1), log(H_e_energy(1)), H_i_part(1), tau_e(1),
    tau_i_energy(1), tau_e_energy(1), tau_i_part(1), H98(1), H98(1)*tau_e(1), Q_ie1(1), Q_ie2(1), Q_ie3
    (1), H98_calc, taue_raw(1), tauth_raw(1), taue_raw(1)/tau_e(1), tauth_raw(1)/tau_e(1), taueh98y2_raw
    (1),densv2(1),densv2_calc(1)
167
168         !OUTPUT_GLOBAL2 (USED TO BUILD REGRESSION MODELS FOR H_FACTORS
169         IF (sim_mode.eq.4) Then
170                 Do i=1,array_size1
171                         If (mod(i,1).eq.0.and.t_exp(i).le.(end_time-0.1)) Then
172                                 If (.not.isnan(ni(i)).and.H_i_energy(i).le.1.0.and.H_i_energy
    (i).gt.0) Then
173                                 !If (.not.isnan(ni(i))) Then
174
175                                         pauxeon = 0
176                                         pauxion = 0
177                                         counterfracon = 0
178                                         shortfracall = 1
179                                         gason = 0
180
181                                         if (nbi_counter_frac(i).gt.0) counterfracon = 1
182                                         if (nbi_short_frac(i).lt.1) shortfracall = 0
183                                         if (S_i_gas(i)/volume(i).ge.5.0E17) gason = 1
184
185
186
187                                         Write (4,*)     shot,t_exp(i), &
188                                                         PNBI(i)/volume(i), nbi_counter_frac(i),
    nbi_short_frac(i), &
189                                                         Paux_e_vol(i), Paux_i_vol(i), q95_raw(i),
    q0_raw(i), &
190                                                         IP(i)/xsec_area(i), BT0_abs(i), S_i_gas(i)/
    volume(i)*1E-19, &
191                                                         tri_divert(i), tri_nondivert(i), ELONG_A
    (i), Q_ie(i), &
192                                                         ni(i), Ti_kev(i), Te_kev(i), Ti_kev(i)/
    Te_kev(i), ni_19(i)/(H_i_part(i)*tau_e(i)), &
193                                                         H_i_energy(i), H_e_energy(i),H_i_part
    (i),S_i_rec_vol_tot(i)
194
195                                 End If
196                         End If
197                         If (t_exp(i).ge.end_time) Exit
198                 End Do
199         End If
200
201         !OUTPUT_GLOBAL5 (USED TO GET STATISTICS FOR ALL THE SHOTS)
202         IF (sim_mode.eq.4) Then
203                 out_count=0
204                 Do i=1,array_size1
205                         !If (isnan(ni(i)).or.isnan(ne(i)).or.isnan(Ti_keV(i)).or.isnan(Te_keV(i)))
    Exit
206                         If (t_exp(i).ge.start_time.and.t_exp(i).le.end_time.and.t_sim(i).ge.200)
    Then
207                         !If (t_exp(i).ge.start_time.and.t_exp(i).le.start_time+5.0) Then
208                                 If(t_sim(i).ge.out_count*50+200) Then
209                                         If (isnan(ni(i))) Then
210                                                 !do nothing
211                                         Else
```

```
212                                                 Write (19,*)    shot,t_exp(i), &
213                                                                 PNBI(i), nbi_counter_frac(i), &
214                                                                 Paux_e(i), Paux_i(i), q95_raw(i), &
215                                                                 IP(i), BT0_abs(i), S_i_gas(i), &
216                                                                 tri_divert(i), tri_nondivert(i),
    ELONG_A(i), &
217                                                                 ni(i), Ti_keV(i), Te_keV(i)
218                                                 out_count = out_count + 1
219                                         End If
220                                 End If
221                         End If
222                         If (t_exp(i).ge.end_time) Exit
223                 End Do
224         End If
225
226         !output_modes
227         ! 1 = output raw imported timeseries data
228         ! 2 = normal for plotting, use when everything is working
229
230         !OUTFILE2 - GENERAL SHOT OUTPUT
231         If (output_mode.eq.2) Then
232                 Open (unit = 11, File = outfile2)
233                 out_count=0
234                 Write (11,*) 't_exp t_sim H98 H_i_energy H_e_energy H_i_part tau_e ne ni Te_keV
    Ti_keV Te_keV/Ti_keV ne_19_td te_td ti_td IP BT0_abs q95_raw li_raw PNBI POH Paux_e Paux_i GASA_CAL
    BDOTAMPL nu_ne nu_te nu_ti'
235                 Do i=1,i_max
236                         If (isnan(ni(i)).or.isnan(ne(i)).or.isnan(Ti_keV(i)).or.isnan(Te_keV(i)))
    Exit
237                         Write (11,*)    t_exp(i),t_sim(i), H98(i), H_i_energy(i), H_e_energy(i),
    H_i_part(i), tau_e(i), &
238                                         ne(i), ni(i), Te_keV(i), Ti_keV(i), Te_keV(i)/Ti_keV(i), &
239                                         ne_19_td(i), te_td(i), ti_td(i), &
240                                         IP(i), BT0_abs(i), q95_raw(i), li_raw(i), &
241                                         PNBI(i), POH(i), Paux_e(i), Paux_i(i), dat_synced(i,4),
    BDOTAMPL(i), &
242                                         nu_ne(i), nu_te(i), nu_ti(i)
243                 End Do
244                 Close (11)
245         End If
246
247         !OUTFILE4 - COMPARE PREDICTED WITH EXP
248         If (sim_mode.eq.1) Then
249                 Open (unit = 10, File = outfile8)
250                 Open (unit = 9, File = outfile4)
251                 Read (10,*) dummy_char
252                 Write (9,*) 't_exp H_i_part_exp H_i_part H_i_energy_exp H_i_energy H_e_energy_exp
    H_e_energy ni_exp ni Ti_kev_exp Ti_kev Te_kev_exp Te_kev'
253                 Do i=1,i_max
254                         If (t_exp(i).gt.(end_time-0.05)) Exit
255
256                         Read (10,*) dummy, H_i_energy_exp, H_e_energy_exp, H_i_part_exp, ni_19_exp,
    ne_19_exp, nz1_19_exp, nz2_19_exp, nz3_19_exp, nz4_19_exp, nz5_19_exp, nz6_19_exp,
    nz7_19_exp,nz8_19_exp, nz9_19_exp, nz10_19_exp, Ti_keV_exp, Te_keV_exp, nu_ne_exp, nu_Ti_exp,
    nu_Te_exp
257
258                         !If (isnan(ni_19_exp).or.isnan(Ti_kev_exp).or.isnan(Te_kev_exp).or.isnan
    (ni_19(i))) Exit
259
260                         ni_19_exp_array(i) = ni_19_exp
261                         Ti_keV_exp_array(i) = Ti_keV_exp
262                         Te_keV_exp_array(i) = Te_keV_exp
263
264                         Write (9,*) t_exp(i), H_i_part_exp, H_i_part(i), H_i_energy_exp, H_i_energy
    (i), H_e_energy_exp, H_e_energy(i), ni_19_exp*1E-19, ni(i), Ti_kev_exp, Ti_kev(i), Te_kev_exp,
    Te_kev(i)
265                 End Do
266                 Close (10)
267                 Close (9)
268         End If
```

```
269
270          !OUTPUT RADIAL TIMESERIES DATA FOR PLOTTING
271          If (sim_mode.eq.4) Then
272                  Open (unit = 18, File = outfile9)
273                  Write (18,*) 'ne_time ne_0_seg ne_5_seg ne_9_seg Ti_time Ti_0_seg Ti_5_seg Ti_9_seg
     Te_time Te_0_seg Te_5_seg Te_9_seg'
274                  Do i=1,array_size7
275                          If (Te_time(i).gt.0) Then
276                                  Write (18,*) ne_time(i), ne_0_seg(i), ne_5_seg(i), ne_9_seg(i),
     Ti_time(i), Ti_0_seg(i), Ti_5_seg(i), Ti_9_seg(i), Te_time(i), Te_0_seg(i), Te_5_seg(i), Te_9_seg(i)
277                          End If
278                  End Do
279                  Close (18)
280          End If
281          !GOODNESS OF FIT CALCULATION
282          If (sim_mode.eq.1) Then
283                  gof_width = 0.2
284                  t_start_gof = start_time
285                  t_end_gof = t_start_gof + gof_width
286
287                  Do i_gof = 1,100
288                          If (t_end_gof.le.actual_end_time-0.1) Then
289
290
291                                  ni_gof(i_gof)  = ABS(sum(ni_19,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)/count
     (t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof) - &
292                                                  sum(ni_19_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)/count
     (t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)) / &
293                                                  (sum(ni_19_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)/count
     (t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof))
294
295                                  Te_gof(i_gof)  = ABS(sum(Te_keV,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)/count
     (t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof) - &
296                                                  sum(Te_keV_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)/count
     (t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)) / &
297                                                  (sum(Te_keV_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)/count
     (t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof))
298
299                                  Ti_gof(i_gof)  = ABS(sum(Ti_keV,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)/count
     (t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof) - &
300                                                  sum(Ti_keV_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)/count
     (t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)) / &
301                                                  (sum(Ti_keV_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof)/count
     (t_exp.gt.t_start_gof.and.t_exp.le.t_end_gof))
302
303
304
305                                  If (isnan(ni_gof(i_gof)).or.isnan(ni_gof(i_gof)).or.isnan(ni_gof
     (i_gof))) Then
306                                          ni_gof(i_gof) = 0
307                                          Ti_gof(i_gof) = 0
308                                          Te_gof(i_gof) = 0
309                                  End If
310
311                                  !Print *,i_gof,ni_gof(i_gof), Ti_gof(i_gof), Te_gof(i_gof)
312
313                                  t_start_gof = t_end_gof
314                                  t_end_gof = t_start_gof + gof_width
315                          Else
316                          !       ni_gof(i_gof)  = ABS(sum(ni_19,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.actual_end_time)/count
```

```
     (t_exp.gt.t_start_gof.and.t_exp.lt.actual_end_time) - &
317                          !                          sum(ni_19_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.actual_end_time)/count
     (t_exp.gt.t_start_gof.and.t_exp.lt.actual_end_time)) / &
318                          !                          (sum(ni_19_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.actual_end_time)/count
     (t_exp.gt.t_start_gof.and.t_exp.lt.actual_end_time))
319
320                          !       Ti_gof(i_gof)  = ABS(sum(Ti_kev,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.actual_end_time)/count
     (t_exp.gt.t_start_gof.and.t_exp.lt.actual_end_time) - &
321                          !                          sum(Ti_kev_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.actual_end_time)/count
     (t_exp.gt.t_start_gof.and.t_exp.lt.actual_end_time)) / &
322                          !                          (sum(Ti_kev_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.actual_end_time)/count
     (t_exp.gt.t_start_gof.and.t_exp.lt.actual_end_time))
323
324                          !       Te_gof(i_gof)  = ABS(sum(Te_kev,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.actual_end_time)/count
     (t_exp.gt.t_start_gof.and.t_exp.lt.actual_end_time) - &
325                          !                          sum(Te_kev_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.actual_end_time)/count
     (t_exp.gt.t_start_gof.and.t_exp.lt.actual_end_time)) / &
326                          !                          (sum(Te_kev_exp_array,
     mask=t_exp.gt.t_start_gof.and.t_exp.le.actual_end_time)/count
     (t_exp.gt.t_start_gof.and.t_exp.lt.actual_end_time))
327
328                                  Exit
329                          End If
330
331                 End Do
332
333                 ni_gof_av = sum(ni_gof)/(max(1,count(ni_gof.gt.0)))
334                 Ti_gof_av = sum(Ti_gof)/(max(1,count(Ti_gof.gt.0)))
335                 Te_gof_av = sum(Te_gof)/(max(1,count(Te_gof.gt.0)))
336
337                 Write (17,*) shot, ni_gof_av, Ti_gof_av, Te_gof_av
338          End If
339
340
341          !OUTFILE_GLOBAL6 - RANGES OF PARAMETERS
342          Write (21,*)    shot, start_time, end_time, &
343                          minval(Paux_e,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
344                          maxval(Paux_e,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
345                          minval(Paux_i,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
346                          maxval(Paux_i,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
347                          minval(GASA_CAL,mask=t_exp.gt.start_time.and.t_exp.le.end_time)/18.27417, &
348                          maxval(GASA_CAL,mask=t_exp.gt.start_time.and.t_exp.le.end_time)/18.27417, &
349                          minval(PNBI,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
350                          maxval(PNBI,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
351                          minval(BT0_abs,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
352                          maxval(BT0_abs,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
353                          minval(q95_raw,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
354                          maxval(q95_raw,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
355                          minval(ni,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
356                          maxval(ni,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
357                          minval(Te_keV,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
358                          maxval(Te_keV,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
359                          minval(Ti_keV,mask=t_exp.gt.start_time.and.t_exp.le.end_time), &
360                          maxval(Ti_keV,mask=t_exp.gt.start_time.and.t_exp.le.end_time)
361
362
363
364   End Do
365   Close (3)
366   Close (4)
367   Close (16)
368   Close (17)
369
```

```
370   End Program GT_BURN2
```

```
Subroutine Allocate_arrays
Use Variables
Implicit NONE


!Allocate array sizes
array_size1=300000 !Main timeseries output from simulation
array_size2=50 !Radial points in synthesized radial distribution
array_size3=500000 !Stuff for Fast Ion calculations
array_size4=10 !number of impurity populations
array_size5=100 !number of possible shots to read in
array_size6=33 !number of timeseries data points being read in
array_size7=24 !number of segments and Ti data points
array_size8=33 !number of timeseries datasets that are used for pre/post event comparisons
array_size9=100 !maximum number of windows for goodness of fit calculation

Allocate(dat(array_size1,array_size6))
dat = 0.0d0
Allocate(dat_synced_td(array_size1,array_size6))
dat_synced_td = 0.0d0
Allocate(dat_smooth(array_size1,array_size6))
dat_smooth = 0.0d0
Allocate(dat_time(array_size1,array_size6))
dat_time = 0.0d0
Allocate(start_point(array_size6))
start_point = 0.0d0
Allocate(loc_prev(array_size6))
loc_prev = 0.0d0
Allocate(loc_after(array_size6))
loc_after = 0.0d0
Allocate(dat_synced(array_size1,array_size6))
dat_synced = 0.0d0
Allocate(dat_synced_smooth(array_size1,array_size6))
dat_synced_smooth = 0.0d0
Allocate(dat_synced_clean(array_size1,array_size6))
dat_synced_clean = 0.0d0
Allocate(data_td(array_size1,array_size6))
data_td = 0.0d0
Allocate(initial_av(array_size6))
initial_av = 0.0d0
Allocate(start_value_td(array_size6))
start_value_td = 0.0d0
Allocate(end_value_td(array_size6))
end_value_td = 0.0d0
Allocate(av_time(array_size1))
av_time = 0.0d0
Allocate(av_data(array_size1))
av_data = 0.0d0
Allocate(t_sim(array_size1))
t_sim = 0.0d0
Allocate(t_exp(array_size1))
t_exp = 0.0d0
Allocate(dat_temp(array_size1))
dat_temp = 0.0d0
Allocate(dat_time_temp(array_size1))
dat_time_temp = 0.0d0
Allocate(H98(array_size1))
H98 = 0.0d0
Allocate(IP(array_size1))
IP = 0.0d0
Allocate(BT0(array_size1))
BT0 = 0.0d0
Allocate(GASA_CAL(array_size1))
GASA_CAL = 0.0d0
Allocate(PNBI(array_size1))
PNBI = 0.0d0
Allocate(POH(array_size1))
POH = 0.0d0
Allocate(POH_vol(array_size1))
POH_vol = 0.0d0
```

```
Subroutine data_smooth
Use Variables
Implicit NONE

!Called by 'timeseries'

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! RAW DATA SMOOTH
! Will eventually be done with an SGOLAY routing. For now, it's just a moving average
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!Read in data and close file
Open (unit = 7, File = datafile)
Read (7,*) max_pts !The first line is the total number of points in the longest data series
Print *,'max_pts = ',max_pts
Do i = 1,max_pts
        Read (7,*) (dat_time(i,k), dat(i,k), k=1,array_size6)
End Do
Close (7)
dat_time = dat_time*1E-3 !all raw times now in seconds instead of ms

!Clean up data a little bit
Do k=1,array_size6
        Do i = 1,max_pts
                If (isnan(dat(i,k))) dat(i,k)=0.0
        End Do
End Do

!Smooth Raw data (moving average, width specified at top of 'timeseries.f95' or maybe in an input file later
Do k=1,array_size6
        Print *,'k = ',k
        Do i = 1,max_pts
                dat_temp(i) = dat(i,k)
                dat_time_temp(i) = dat_time(i,k)
        End Do
        Do i = 1,max_pts
                If (dat_time(i,k).eq.0.and.dat_time(i+10,k).eq.0) Exit
                If (dat_time(i,k).gt.end_time) Exit

                If ((dat_time(i,k)-dat_time(1,k)).le.smooth_window) Then
                        dat_smooth(i,k) = sum(dat_temp, mask=dat_time_temp.le.dat_time(i,k))/count(dat_time_temp.le.dat_time(i,k))
                Else
                        dat_smooth(i,k) = sum(dat_temp, mask=dat_time_temp.ge.(dat_time(i,k)-smooth_window).and.dat_time_temp.le.(dat_time(i,k)+smooth_window)) / &
                                          count(dat_time_temp.ge.(dat_time(i,k)-smooth_window).and.dat_time_temp.le.(dat_time(i,k)+smooth_window))
                End If
        End Do
End Do

!Clean up smoothed data of any NaN's
Do k=1,array_size6
        Do i = 1,max_pts
                If (isnan(dat_smooth(i,k))) dat_smooth(i,k)=200000
        End Do
End Do

Print *,'writing smoothed raw data to file'
!Write to output to save time
Open (unit = 20, File = datafile_smooth)
Write (20,*) max_pts, smooth_window
Do i=1,max_pts
        Write (20,*) (dat_time(i,k), dat_smooth(i,k), k=1,array_size6)
End Do
Close (20)

End Subroutine
```

```
Subroutine f2e_prep_timeseries
Use Variables
Implicit NONE
EXTERNAL DGELS

!New approach
!	Construct a time profile of x_9, x_5, and x_0 and calculate a volume, density, averaged quantity from those
!	time profiles at each time step. This is definitely the most accurate way of doing it, but also the most
!	time intensive because until I know how to automatically process radial data from D3D, I have to grab about
!	3 radial points for about 20 time points for each quantity (n, Ti, Te) for each shot.

Print *,'Building density and temperature time profiles'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Create synced x_0, x_5, and x_9 time profiles
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Do i=1,array_size1
	If (t_exp(i).gt.end_time) Exit

	!for each t_exp, find the positions of the dat_times just less than and just greater than t_exp
	loc_prev_ne = maxloc(ne_time,dim=1,mask=ne_time.le.t_exp(i))
	loc_after_ne = loc_prev_ne+1
	loc_prev_Te = maxloc(Te_time,dim=1,mask=Te_time.le.t_exp(i))
	loc_after_Te = loc_prev_Te+1
	loc_prev_Ti = maxloc(Ti_time,dim=1,mask=Ti_time.le.t_exp(i))
	loc_after_Ti = loc_prev_Ti+1

	!ne
	If (ne_time(loc_prev_ne).eq.t_exp(i)) Then !t_exp lines up perfectly with a raw data point
		ne_0(i) = ne_0_seg(loc_prev_ne)
		ne_5(i) = ne_5_seg(loc_prev_ne)
		ne_9(i) = ne_9_seg(loc_prev_ne)
	Else
		ne_0(i) = ne_0_seg(loc_prev_ne) + (ne_0_seg(loc_after_ne)-ne_0_seg(loc_prev_ne))/(ne_time(loc_after_ne)-ne_time(loc_prev_ne)) * (t_exp(i)-ne_time(loc_prev_ne))
		ne_5(i) = ne_5_seg(loc_prev_ne) + (ne_5_seg(loc_after_ne)-ne_5_seg(loc_prev_ne))/(ne_time(loc_after_ne)-ne_time(loc_prev_ne)) * (t_exp(i)-ne_time(loc_prev_ne))
		ne_9(i) = ne_9_seg(loc_prev_ne) + (ne_9_seg(loc_after_ne)-ne_9_seg(loc_prev_ne))/(ne_time(loc_after_ne)-ne_time(loc_prev_ne)) * (t_exp(i)-ne_time(loc_prev_ne))
	End If
	nu_ne(i) = log((ne_5(i)-ne_9(i))/(ne_0(i)-ne_9(i)))/log(1.0-(a_minor(i)/(2.0*(a_minor(i) - delta_ITB_ne)))**2.0)

	!Te
	If (Te_time(loc_prev_Te).eq.t_exp(i)) Then !t_exp lines up perfectly with a raw data point
		Te_0(i) = Te_0_seg(loc_prev_Te)
		Te_5(i) = Te_5_seg(loc_prev_Te)
		Te_9(i) = Te_9_seg(loc_prev_Te)
	Else
		Te_0(i) = Te_0_seg(loc_prev_Te) + (Te_0_seg(loc_after_Te)-Te_0_seg(loc_prev_Te))/(Te_time(loc_after_Te)-Te_time(loc_prev_Te)) * (t_exp(i)-Te_time(loc_prev_Te))
		Te_5(i) = Te_5_seg(loc_prev_Te) + (Te_5_seg(loc_after_Te)-Te_5_seg(loc_prev_Te))/(Te_time(loc_after_Te)-Te_time(loc_prev_Te)) * (t_exp(i)-Te_time(loc_prev_Te))
		Te_9(i) = Te_9_seg(loc_prev_Te) + (Te_9_seg(loc_after_Te)-Te_9_seg(loc_prev_Te))/(Te_time(loc_after_Te)-Te_time(loc_prev_Te)) * (t_exp(i)-Te_time(loc_prev_Te))
	End If
	nu_Te(i) = log((Te_5(i)-Te_9(i))/(Te_0(i)-Te_9(i)))/log(1.0-(a_minor(i)/(2.0*(a_minor(i) - delta_ITB_Te)))**2.0)

	!Ti
	If (Ti_time(loc_prev_Ti).eq.t_exp(i)) Then !t_exp lines up perfectly with a raw data point
		Ti_0(i) = Ti_0_seg(loc_prev_Ti)
		Ti_5(i) = Ti_5_seg(loc_prev_Ti)
		Ti_9(i) = Ti_9_seg(loc_prev_Ti)
	Else
		Ti_0(i) = Ti_0_seg(loc_prev_Ti) + (Ti_0_seg(loc_after_Ti)-Ti_0_seg(loc_prev_Ti))/(Ti_time(loc_after_Ti)-Ti_time(loc_prev_Ti)) * (t_exp(i)-Ti_time(loc_prev_Ti))
```

```
		Ti_5(i) = Ti_5_seg(loc_prev_Ti) + (Ti_5_seg(loc_after_Ti)-Ti_5_seg(loc_prev_Ti))/(Ti_time(loc_after_Ti)-Ti_time(loc_prev_Ti)) * (t_exp(i)-Ti_time(loc_prev_Ti))
		Ti_9(i) = Ti_9_seg(loc_prev_Ti) + (Ti_9_seg(loc_after_Ti)-Ti_9_seg(loc_prev_Ti))/(Ti_time(loc_after_Ti)-Ti_time(loc_prev_Ti)) * (t_exp(i)-Ti_time(loc_prev_Ti))
	End If
	nu_Ti(i) = log((Ti_5(i)-Ti_9(i))/(Ti_0(i)-Ti_9(i)))/log(1.0-(a_minor(i)/(2.0*(a_minor(i) - delta_ITB_Ti)))**2.0)
End Do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!SYNTHESIZE RADIAL DISTRIBUTIONS AND CALCULATE AVERAGED QUANTITIES AT EACH TIME STEP
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Do i=1,array_size1
	If (t_exp(i).gt.end_time) Exit

	delta_ITB_ne = 0.1*a_minor(i)
	delta_ITB_Ti = 0.1*a_minor(i)
	delta_ITB_Te = 0.1*a_minor(i)
	delta_r = a_minor(i)/(rad_pts-1)

	Do j = 1,rad_pts

		!GENERAL (RADIAL GRID AND TOROIDAL SHELL VOLUME CALCULATIONS)
		rval(j) = (j-1)*delta_r
		diff_vol(j) = 2.0*elong_a(i)*pi**2.0*R0(i)*((j*delta_r)**2.0 - ((j-1)*delta_r)**2.0)

		!DENSITY DISTRIBUTION
		If (rval(j).lt.(a_minor(i)-delta_ITB_ne)) Then
			ne_rad(j) = ((ne_0(i) - ne_9(i)) * (1.0 - rval(j)**2.0/(a_minor(i) - delta_ITB_ne)**2)**nu_ne(i) + ne_9(i))
			nz3_rad(j) = ne_rad(j) / (35.0)
			ni_rad(j) = ne_rad(j) - 6.0*nz3_rad(j)
		Else
			ne_rad(j) = ne_9(i) *(a_minor(i) - rval(j)) / delta_ITB_ne
			nz3_rad(j) = ne_rad(j) / (35.0)
			ni_rad(j) = ne_rad(j) - 6.0*nz3_rad(j)
		End If

		If (j.eq.1) ni_0(i) = ni_rad(j)

		!ION TEMPERATURE DISTRIBUTION
		If (rval(j).lt.(a_minor(i)-delta_ITB_Ti)) Then
			Ti_rad(j) = ((Ti_0(i) - Ti_9(i)) * (1.0 - rval(j)**2.0/(a_minor(i) - delta_ITB_Ti)**2)**nu_Ti(i) + Ti_9(i))
			Ti_J_rad(j)   = Ti_rad(j) * e * 1000.0
		Else
			Ti_rad(j) = Ti_9(i) *(a_minor(i) - rval(j)) / delta_ITB_Ti
			Ti_J_rad(j)   = Ti_rad(j) * e * 1000.0
		End If

		!ELECTRON TEMPERATURE DISTRIBUTION
		If (rval(j).lt.(a_minor(i)-delta_ITB_Te)) Then
			Te_rad(j) = ((Te_0(i) - Te_9(i)) * (1.0 - rval(j)**2.0/(a_minor(i) - delta_ITB_Te)**2)**nu_Te(i) + Te_9(i))
			Te_J_rad(j)   = Te_rad(j) * e * 1000.0
		Else
			Te_rad(j) = Te_9(i) *(a_minor(i) - rval(j)) / delta_ITB_Te
			Te_J_rad(j)   = Te_rad(j) * e * 1000.0
		End If

		!OTHER QUANTITIES NEEDED TO CALCULATE THE AVERAGES
		ne_width(j)  = delta_r * ne_rad(j)
		ni_width(j)  = delta_r * ni_rad(j)
		nz3_width(j)  = delta_r * nz3_rad(j)
		vol_width(j) = delta_r * diff_vol(j)
		ne_V_width(j) = delta_r * ne_rad(j) * diff_vol(j)
		ni_V_width(j) = delta_r * ni_rad(j) * diff_vol(j)
		nz3_V_width(j) = delta_r * nz3_rad(j) * diff_vol(j)
		nTi_V(j) = delta_r * ni_rad(j) * Ti_rad(j) * diff_vol(j)
```

```
122                 nTe_V(j) = delta_r * ne_rad(j) * Te_rad(j) * diff_vol(j)
123
124                 !COLLISIONAL ENERGY TRANSFER
125                 coul_log_rad(j) = log(12.0*pi*sqrt((8.854*Ti_rad(j)/1.6021)**3.0*1.0E11/ni_rad(j)))
126                 Q_ie_rad(j) =   (ne_rad(j)*ni_rad(j))*(1.6021**4.0)*(m_e_kg_reduced/
    m_D_kg_reduced)*coul_log_rad(j)*(Te_J_rad(j)-Ti_J_rad(j)) / &
127                                 (((2*pi)**(1.5))*(8.854**2.0)*sqrt(m_e_kg)*((Te_J_rad(j))**1.5)) *
    1E-14 / 1E6
128                 If (isnan(Q_ie_rad(j))) Q_ie_rad(j)=0 !happens when Ti=Te, i.e. at rho=a. Force to
    zero.
129                 Q_ie_rad_vol(j) = Q_ie_rad(j) * diff_vol(j)
130         End Do
131
132         !AVERAGED QUANTITIES
133         Q_ie(i) = Q_ie_mult*sum(Q_ie_rad_vol)/sum(diff_vol)
134         ne(i) = SUM(ne_V_width) / SUM(vol_width)
135         ni(i) = SUM(ni_V_width) / SUM(vol_width)
136         nz3(i)  = SUM(nz3_V_width) / SUM(vol_width)
137         Te_keV(i) = SUM(nTe_V) / SUM(ne_V_width)
138         Ti_keV(i) = SUM(nTi_V) / SUM(ni_V_width)
139
140         !IMPURITIES OTHER THAN CARBON (nz3)
141         nz1(i)  = fz(1)*ni(i)/(1-fz_tot)
142         nz2(i)  = fz(2)*ni(i)/(1-fz_tot)
143         nz4(i)  = fz(4)*ni(i)/(1-fz_tot)
144         nz5(i)  = fz(5)*ni(i)/(1-fz_tot)
145         nz6(i)  = fz(6)*ni(i)/(1-fz_tot)
146         nz7(i)  = fz(7)*ni(i)/(1-fz_tot)
147         nz8(i)  = fz(8)*ni(i)/(1-fz_tot)
148         nz9(i)  = fz(9)*ni(i)/(1-fz_tot)
149         nz10(i) = fz(10)*ni(i)/(1-fz_tot)
150
151         Z_eff(i) =      (ni(i)+nz1(i)*z(1)**2+nz2(i)*z(2)**2+nz3(i)*z(3)**2+nz4(i)*z(4)**2+nz5(i)*z
    (5)**2 + &
152                         nz6(i)*z(6)**2+nz7(i)*z(7)**2+nz8(i)*z(8)**2+nz9(i)*z(9)**2+nz10(i)*z
    (10)**2)/ne(i)
153
154         !TIME DERIVITIVES
155         If (i.ge.2) Then
156                 ne_td(i) = (ne(i)-ne(i-1))/delta_t
157                 ni_td(i) = (ni(i)-ni(i-1))/delta_t
158                 nz3_td(i) = (nz3(i)-nz3(i-1))/delta_t
159                 Te_td(i) = (Te_keV(i)-Te_keV(i-1))/delta_t
160                 Ti_td(i) = (Ti_keV(i)-Ti_keV(i-1))/delta_t
161         End If
162         If (i.eq.2) Then
163                 ne_td(1) = ne_td(2)
164                 ni_td(1) = ni_td(2)
165                 nz3_td(1) = nz3_td(2)
166                 Te_td(1) = Te_td(2)
167                 Ti_td(1) = Ti_td(2)
168         End If
169
170         !UNIT CONVERSIONS, ETC.
171         ne_19_td(i) = ne_td(i) * 1E19
172         ne_19(i) = ne(i) * 1E19
173         ni_19_td(i) = ni_td(i) * 1E19
174         ni_19(i) = ni(i) * 1E19
175
176         Te_J(i)   = Te_keV(i) * e * 1000.0
177         Te_MJ(i)  = Te_keV(i) * e / 1000.0
178         Te_MJ_td(i)  = Te_td(i) * e / 1000.0
179
180         Ti_J(i)   = Ti_keV(i) * e * 1000.0
181         Ti_MJ(i)  = Ti_keV(i) * e / 1000.0
182         Ti_MJ_td(i)  = Te_td(i) * e / 1000.0
183
184         nz1_19(i)  = nz1(i) * 1E19
185         nz2_19(i)  = nz2(i) * 1E19
186         nz3_19(i)  = nz3(i) * 1E19
```

```
187           nz4_19(i)  = nz4(i) * 1E19
188           nz5_19(i)  = nz5(i) * 1E19
189           nz6_19(i)  = nz6(i) * 1E19
190           nz7_19(i)  = nz7(i) * 1E19
191           nz8_19(i)  = nz8(i) * 1E19
192           nz9_19(i)  = nz9(i) * 1E19
193           nz10_19(i) = nz10(i) * 1E19
194   End Do
195
196   End Subroutine
```

```
Subroutine fast_ion
Use Variables
Implicit NONE

!NBI electron ion split (only doing once at the beginning of run, will change this later)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! PART 1 - NBI FAST IONS IN DEUTERIUM PLASMA
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

ne_fi = ne_0(i)
ni_fi = ni_0(i)
Te_fi = Te_0(i)
Ti_fi = Ti_0(i)

!Initial Parameters
delta_t_fi = .001
Z_b = 1
wb_0_J = nbi1e(i)*1000*e

Te_j_fi = Te_fi * 1000 * e !takes Te_fi in keV but needs in Joules for the calculation
wb_therm_J = Ti_fi * 1000 * e !assumes thermalized at Ti_fi
coul_log(i) = log(12.0*pi*sqrt((8.854*Ti_fi/1.6021)**3.0*1.0E11/ni_fi))

!Calculate B_fi
!B_fi = -sqrt(2.0) * ne_fi * Z_b**(2.0) * e_reduced**(4.0) * sqrt(m_e_kg_reduced) * coul_log(i) /
(6.0 * pi**(1.5) * eps_0_reduced**(2.0) * m_D_kg_reduced)
!B_fi = B_fi / 3.162277E22
!C_fi = 3*pi**0.5 * m_D_kg**1.5 / (4*sqrt(m_e_kg)*m_D_kg)


!Zero out arrays
Do j_fi=1,array_size3
        Wb(j_fi) = 0
        delta_wb_i(j_fi) = 0
        delta_wb_e(j_fi) = 0
        delta_wb_tot(j_fi) = 0
End Do

!Populate E and delta_wb arrays
nbi_i = 0
nbi_e = 0
Wb(1) = wb_0_J
Do j_fi=1,array_size3

        !delta_wb_i(j_fi) = (B_fi*C_fi/sqrt(Wb(j_fi)))*delta_t_fi
        !delta_wb_e(j_fi) = (B_fi*Wb(j_fi)/wb_therm_J**1.5)*delta_t_fi

        If (Wb(j_fi).lt.wb_therm_J) Exit

        delta_wb_i(j_fi) = -sqrt(2.0)*ni_fi*(e_reduced**4.0)*coul_log(i)/(8*pi*
(eps_0_reduced**2.0)*sqrt(m_D_kg_reduced)*sqrt(Wb(j_fi)))
        delta_wb_i(j_fi) = delta_wb_i(j_fi)/3.162277E19*delta_t_fi

        delta_wb_e(j_fi) = -sqrt(2.0)*ne_fi*(e_reduced**4.0)*sqrt(m_e_kg_reduced)*coul_log(i)*Wb
(j_fi)/(6*(pi**1.5)*(eps_0_reduced**2.0)*m_d_kg_reduced*(Te_j_fi**1.5))
        delta_wb_e(j_fi) = delta_wb_e(j_fi)/3.162277E19*delta_t_fi

        delta_wb_tot(j_fi) = delta_wb_i(j_fi) + delta_wb_e(j_fi)
        Wb(j_fi+1) = Wb(j_fi) + delta_wb_tot(j_fi)

        nbi_i = nbi_i + delta_wb_i(j_fi)
        nbi_e = nbi_e + delta_wb_e(j_fi)

End Do

nbi_i_frac(i) = abs(nbi_i)/(abs(nbi_i)+abs(nbi_e))
nbi_i_frac(i) = nbi_i_frac(i)
nbi_e_frac(i) = 1-nbi_i_frac(i)

! Calculation of steady-state energy distribution
```

```
 68
 69  !Energy bin width stuff
 70  !en_range = wb_0_J - wb_therm_J
 71  !en_bins_fi = 20
 72  !en_bin_width = en_range / (en_bins_fi)
 73
 74  !Assign each value in Wb(j) to a bin
 75  !Do j_fi=1,j_fi_max
 76  !       If (j_fi.eq.1) Then
 77  !               Wb_bin(j_fi) = en_bins_fi !this is just a fix up.
 78  !       Else
 79  !               Wb_bin(j_fi) = INT((Wb(j_fi)-wb_therm_J)/en_bin_width) + 1
 80  !       End If
 81  !End Do
 82
 83  !Count the number in each bin
 84  !Do j_fi=1,en_bins_fi
 85  !       Wb_bin_count(j_fi) = count(Wb_bin.eq.j_fi)
 86  !End Do
 87
 88  !nbi_i_frac(i) = sum(delta_wb_i)/(sum(delta_wb_i)+sum(delta_wb_e)) * 1.0
 89
 90  !If (ni(i).ge.4.0) nbi_i_frac(i) = nbi_i_frac(i) * 1.4
 91  !nbi_e_frac(i) = 1.0-nbi_i_frac(i)
 92
 93  !Print *,'nbi_i_frac = ',nbi_i_frac(i)
 94
 95
 96
 97  !If (sim_mode.eq.1) Then
 98  !       Print *,'nbi_i_frac = ',nbi_i_frac
 99  !       Print *,'nbi_e_frac = ',nbi_e_frac
100  !       Print *,''
101  !End If
102
103
104  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
105  ! PART 2 - FAST ALPHAS IN DEUTERIUM PLASMA
106  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
107
108  alpha_e_frac = 0.2
109  alpha_i_frac = 0.8
110
111  !If (i.eq.1) Then
112  !       Open (unit = 15,FILE = fast_ion_out)
113
114  !       Write (15,*) 't_fi Wb(j_fi) delta_wb_i(j_fi) delta_wb_e(j_fi) delta_wb_tot(j_fi) wb_therm_J'
115  !       Do j_fi = 1,j_fi_max
116  !               Write (15,*) j_fi*delta_t_fi, Wb(j_fi), -1*delta_wb_i(j_fi), -1*delta_wb_e(j_fi),
     -1*delta_wb_tot(j_fi), wb_therm_J
117  !       End Do
118  !       Close (15)
119  !End If
120
121  End Subroutine
```

```
Subroutine initial_n_t
Use Variables
Implicit NONE

i=1

! 1. Sort through timeseries-radial data to find the values for *_0, *_5, and *_9 that bracket the desired start time
! 2. Interpolate to find the values for *_0, *_5, and *_9 that correspond to the desired start time
! 3. Calculate the initial densities and temperatures
! 4. Calculate initial Q_ie if necessary

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! DENSITY ne_0, ne_9, nu_ne
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Do m=1,array_size7
        If (ne(m+1).gt.t_exp(i)) Exit
End Do
If (ne(m).eq.t_exp(i)) Then
        ne_0(i) = ne_0_seg(m)
        ne_5(i) = ne_5_seg(m)
        ne_9(i) = ne_9_seg(m)
        nu_ne(i) = log((ne_5(i)-ne_9(i))/(ne_0(i)-ne_9(i)))/log(1-(a_minor(i)/(2.0*(a_minor(i) - delta_ITB_ne)))**2.0)
        ! Calculate quantity at each segment time
Else
        ne_0(i) = (ne_0_seg(m+1)-ne_0_seg(m))/(ne(m+1)-ne(m)) * (t_exp(i)-ne(m)) + ne_0_seg(m)
        ne_5(i) = (ne_5_seg(m+1)-ne_5_seg(m))/(ne(m+1)-ne(m)) * (t_exp(i)-ne(m)) + ne_5_seg(m)
        ne_9(i) = (ne_9_seg(m+1)-ne_9_seg(m))/(ne(m+1)-ne(m)) * (t_exp(i)-ne(m)) + ne_9_seg(m)
        nu_ne(i) = log((ne_5(i)-ne_9(i))/(ne_0(i)-ne_9(i)))/log(1-(a_minor(i)/(2.0*(a_minor(i) - delta_ITB_ne)))**2.0)
End If

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! ION TEMPERATURE Ti_0, Ti_9, nu_Ti
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Do m=1,array_size7
        If (Ti(m+1).gt.t_exp(i)) Exit
End Do
If (Ti(m).eq.t_exp(i)) Then
        Ti_0(i) = Ti_0_seg(m)
        Ti_5(i) = Ti_5_seg(m)
        Ti_9(i) = Ti_9_seg(m)
        nu_Ti(i) = log((Ti_5(i)-Ti_9(i))/(Ti_0(i)-Ti_9(i)))/log(1-(a_minor(i)/(2.0*(a_minor(i) - delta_ITB_Ti)))**2.0)
        !Calculate averaged quantity at each segment time
Else
        Ti_0(i) = (Ti_0_seg(m+1)-Ti_0_seg(m))/(Ti(m+1)-Ti(m)) * (t_exp(i)-Ti(m)) + Ti_0_seg(m)
        Ti_5(i) = (Ti_5_seg(m+1)-Ti_5_seg(m))/(Ti(m+1)-Ti(m)) * (t_exp(i)-Ti(m)) + Ti_5_seg(m)
        Ti_9(i) = (Ti_9_seg(m+1)-Ti_9_seg(m))/(Ti(m+1)-Ti(m)) * (t_exp(i)-Ti(m)) + Ti_9_seg(m)
        nu_Ti(i) = log((Ti_5(i)-Ti_9(i))/(Ti_0(i)-Ti_9(i)))/log(1-(a_minor(i)/(2.0*(a_minor(i) - delta_ITB_Ti)))**2.0)
End If

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! ELECTRON TEMPERATURE Te_0, Te_9, nu_Te
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Do m=1,array_size7
        If (Te(m+1).gt.t_exp(i)) Exit
End Do
If (Te(m).eq.t_exp(i)) Then
        Te_0(i) = Te_0_seg(m)
        Te_5(i) = Te_5_seg(m)
        Te_9(i) = Te_9_seg(m)
        nu_Te(i) = log((Te_5(i)-Te_9(i))/(Te_0(i)-Te_9(i)))/log(1-(a_minor(i)/(2.0*(a_minor(i) - delta_ITB_Te)))**2.0)
        !Calculate averaged quantity at each segment time
Else
        Te_0(i) = (Te_0_seg(m+1)-Te_0_seg(m))/(Te(m+1)-Te(m)) * (t_exp(i)-Te(m)) + Te_0_seg(m)
        Te_5(i) = (Te_5_seg(m+1)-Te_5_seg(m))/(Te(m+1)-Te(m)) * (t_exp(i)-Te(m)) + Te_5_seg(m)
```

```
65          Te_9(i) = (Te_9_seg(m+1)-Te_9_seg(m))/(Te(m+1)-Te(m)) * (t_exp(i)-Te(m)) + Te_9_seg(m)
66          nu_Te(i) = log((Te_5(i)-Te_9(i))/(Te_0(i)-Te_9(i)))/log(1-(a_minor(i)/(2.0*(a_minor(i) -
    delta_ITB_Te)))**2.0)
67  End If
68
69  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
70  ! CONSTRUCT RADIAL PROFILES AND CALCULATE THINGS FOR EACH TIME STEP
71  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
72  m_te=1
73  m_ti=1
74  m_ne=1
75  !Do i=1,array_size1
76          delta_ITB_ne = 0.1*a_minor(i)
77          delta_ITB_Ti = 0.1*a_minor(i)
78          delta_ITB_Te = 0.1*a_minor(i)
79          delta_r = a_minor(i)/(rad_pts-1)
80
81          !SYNTHESIZE RADIAL DISTRIBUTIONS
82          Do j = 1,rad_pts
83                  !GENERAL
84                  rval(j) = (j-1)*delta_r
85                  diff_vol(j) = 2.0*elong_a(i)*pi**2.0*R0(i)*((j*delta_r)**2.0 - ((j-1)*delta_r)**2.0)
86                  diff_vol_frac(j) = diff_vol(j)/(2.0*elong_a(i)*pi**2.0*a_minor(i)**2.0*R0(i))
87                  !DENSITY
88
89                  If (rval(j).lt.(a_minor(i)-delta_ITB_ne)) Then
90                          ne_rad(j) = ((ne_0(i) - ne_9(i)) * (1.0 - rval(j)**2.0/(a_minor(i) -
    delta_ITB_ne)**2)**nu_ne(i) + ne_9(i))
91                          nz3_rad(j) = ne_rad(j) / (35.0)
92                          ni_rad(j) = ne_rad(j) - 6.0*nz3_rad(j)
93                  Else
94                          ne_rad(j) = ne_9(i) *(a_minor(i) - rval(j)) / delta_ITB_ne
95                          nz3_rad(j) = ne_rad(j) / (35.0)
96                          ni_rad(j) = ne_rad(j) - 6.0*nz3_rad(j)
97                  End If
98
99                  !If (t_exp(i).ge.2000.and.mod(i,10000).eq.0) Print *,'ne2',t_exp(i),ne_0(i),ne_9
    (i),j,ne_rad(j)
100
101                 ne_width(j)  = delta_r * ne_rad(j)
102                 ni_width(j)  = delta_r * ni_rad(j)
103                 nz3_width(j)  = delta_r * nz3_rad(j)
104                 vol_width(j) = delta_r * diff_vol(j)
105                 ne_V_width(j) = delta_r * ne_rad(j) * diff_vol(j)
106                 ni_V_width(j) = delta_r * ni_rad(j) * diff_vol(j)
107                 nz3_V_width(j) = delta_r * nz3_rad(j) * diff_vol(j)
108                 !ION TEMPERATURE
109
110                 If (rval(j).lt.(a_minor(i)-delta_ITB_Ti)) Then
111                         Ti_rad(j) = ((Ti_0(i) - Ti_9(i)) * (1.0 - rval(j)**2.0/(a_minor(i) -
    delta_ITB_Ti)**2)**nu_Ti(i) + Ti_9(i))
112                         Ti_J_rad(j)   = Ti_rad(j) * e * 1000.0
113                 Else
114                         Ti_rad(j) = Ti_9(i) *(a_minor(i) - rval(j)) / delta_ITB_Ti
115                         Ti_J_rad(j)   = Ti_rad(j) * e * 1000.0
116                 End If
117
118                 nTi_V(j) = delta_r * ni_rad(j) * Ti_rad(j) * diff_vol(j)
119                 !ELECTRON TEMPERATURE
120                 If (rval(j).lt.(a_minor(i)-delta_ITB_Te)) Then
121                         Te_rad(j) = ((Te_0(i) - Te_9(i)) * (1.0 - rval(j)**2.0/(a_minor(i) -
    delta_ITB_Te)**2)**nu_Te(i) + Te_9(i))
122                         Te_J_rad(j)   = Te_rad(j) * e * 1000.0
123                 Else
124                         Te_rad(j) = Te_9(i) *(a_minor(i) - rval(j)) / delta_ITB_Te
125                         Te_J_rad(j)   = Te_rad(j) * e * 1000.0
126                 End If
127
128                 nTe_V(j) = delta_r * ne_rad(j) * Te_rad(j) * diff_vol(j)
129                 !COLLISIONAL ENERGY TRANSFER
```

```
130                 coul_log_rad(j) = log(12.0*pi*sqrt((8.854*Ti_rad(j)/1.6021)**3.0*1.0E11/ni_rad(j)))
131                 Q_ie_rad(j) =   (ne_rad(j)*ni_rad(j))*(1.6021**4.0)*(m_e_kg_reduced/
    m_D_kg_reduced)*coul_log_rad(j)*(Te_J_rad(j)-Ti_J_rad(j)) / &
132                                 (((2*pi)**(1.5))*(8.854**2.0)*sqrt(m_e_kg)*((Te_J_rad(j))**1.5)) *
    1E-14 / 1E6
133                 Q_ie_rad_vol(j) = Q_ie_rad(j) * diff_vol(j)
134         End Do
135
136         Q_ie_rad(rad_pts) = 0 !was getting an NaN at rho=1 for Q_ie_rad. It's zero because the
    temperatures are the same. Forcing to zero.
137         Q_ie_rad_vol(rad_pts) = 0 !was getting an NaN at rho=1 for Q_ie_rad. It's zero because the
    temperatures are the same. Forcing to zero.
138         If (Q_ie_mult.eq.0) THen
139                 Q_ie(i) = 0
140         Else
141                 Q_ie(i) = Q_ie_mult*sum(Q_ie_rad_vol)/sum(diff_vol)
142         End If
143
144         !If t_exp is equal to (or just passing) a segment time for ne, Te, or Ti, calculate the
    averaged quantity (Ti_seg, etc.) at that point.
145         !If (t_exp(i).ge.ne(m_ne)) Then
146                 ne(i) = SUM(ne_V_width) / SUM(vol_width)
147                 ni(i) = SUM(ni_V_width) / SUM(vol_width)
148                 nz3(i)  = SUM(nz3_V_width) / SUM(vol_width)
149         !       m_ne = m_ne + 1
150         !End If
151
152         !If (t_exp(i).ge.Te(m_te)) Then
153                 Te_keV(i) = SUM(nTe_V) / SUM(ne_V_width)
154         !       m_te = m_te + 1
155         !End If
156
157         !If (t_exp(i).ge.Ti(m_ti)) Then
158                 Ti_keV(i) = SUM(nTi_V) / SUM(ni_V_width)
159         !       m_ti = m_ti + 1
160         !       !Do j = 1,rad_pts
161         !       !       Print *,i,t_Exp(i),nTi_V(j),ni_V_width(j),ni_rad(j),Ti_rad(j),diff_vol(j),ne
    (i)
162         !       !End Do
163         !End If
164
165         !If (t_exp(i).ge.end_time) Exit
166 !End Do
167
168         Te_MJ(i) = Te_keV(i)*e/1000
169         Ti_MJ(i) = Ti_keV(i)*e/1000
170
171         Te_J(i) = Te_MJ(i) * 1.0E6
172         Ti_J(i) = Ti_MJ(i) * 1.0E6
173
174         ni_19(i) = ni(i)*1E19
175
176         nz1_19(i) = nz1(i)*1E19
177         nz1_19(i) = nz2(i)*1E19
178         nz1_19(i) = nz3(i)*1E19
179         nz1_19(i) = nz4(i)*1E19
180         nz1_19(i) = nz5(i)*1E19
181         nz1_19(i) = nz6(i)*1E19
182         nz1_19(i) = nz7(i)*1E19
183         nz1_19(i) = nz8(i)*1E19
184         nz1_19(i) = nz9(i)*1E19
185         nz1_19(i) = nz10(i)*1E19
186
187         ne_19(i) = ne(i)*1E19
188
189         Ti_Te(i) = Ti_keV(i)/Te_kev(i)
190
191 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
192 ! OUTPUT
193 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
194
195   !Radial output
196   Open (unit = 8, File = rad_outfile)
197   Write (8,*) 'i delta_r rval ne_rad nz3_rad ni_rad Te_rad Ti_rad diff_vol vol_tot coul_log_rad
      Q_ie_rad chrd_wt'
198   Do j=1,rad_pts
199           Write (8,*) i, delta_r, rval(j), ne_rad(j), nz3_rad(j), ni_rad(j), Te_rad(j), Ti_rad(j),
      diff_vol(j), vol_tot, coul_log_rad(j), Q_ie_rad(j), chrd_wt(j)
200   End Do
201   Close (20)
202
203   End Subroutine
```

```
Subroutine radial
Use Variables
Implicit NONE

!New approach
! ne, Te, and Ti timeseries data can each be constructed in one of two ways.
!   1) Calculated the difference between the averaged quantity and the initial value of either DENSV2 or TSTE_TAN
!      and hold that difference constant throughout the shot.
!   2) Construct a time profile of x_9, x_5, and x_0 and calculate a volume, density, averaged quantity from those
!      time profiles at each time step. This is definitely the most accurate way of doing it, but also the most
!      time intensive because until I know how to automatically process radial data from D3D, I have to grab about
!      3 radial points for about 20 time points for each quantity (n, Ti, Te) for each shot.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! DENSITY
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Set up radial geometry
vol_tot_flat = 0
delta_ITB_ne = 0.1*a_minor(i)
delta_ITB_Te = 0.1*a_minor(i)
delta_ITB_Ti = 0.1*a_minor(i)
Do j = 1,rad_pts
        delta_r = a_minor(i)/(rad_pts-1)
        rval(j) = (j-1)*delta_r
        diff_vol(j) = 2.0*elong_a(i)*pi**2.0*R0(i)*((j*delta_r)**2.0 - ((j-1)*delta_r)**2.0)
        diff_vol(j) = sqrt(diff_vol(j))
        vol_tot_flat = vol_tot + diff_vol(j)
        !diff_vol_frac(j) = diff_vol(j)/(2.0*elong_a(i)*pi**2.0*a_minor(i)**2.0*R0(i))
End Do

Do j = 1,rad_pts
        diff_vol_frac(j) = diff_vol(j)/vol_tot_flat
End Do

If (i.ge.2.and.h_reuse.eq.0) Then
        nu_ne(i) = nu_ne(i-1)
        nu_Te(i) = nu_ne(i-1)
        nu_Ti(i) = nu_ne(i-1)
End If

!Calculate ne_0 and ne_9. Hold nu_ne constant so ne_5 not necessary
A1_ne = 0
A2_ne = 0
A3_ne = 0
Do j=1,rad_pts-1
        If (rval(j).lt.(a_minor(i)-delta_ITB_ne)) Then
                A1_ne = A1_ne + diff_vol(j)*((gas_ne_C1*gasa_cal(i)+gas_ne_C2)*(1.0-(rval(j)/(a_minor(i)-delta_ITB_ne))**2.0)**nu_ne(i) + 1.0)
                !A1_ne = A1_ne + diff_vol(j)*(((ne_0-ne_9)/ne_9)*(1.0-(rval(j)/(a_minor(i)-delta_ITB_ne))**2.0)**nu_ne + 1.0)
        Else
                A2_ne = A2_ne + diff_vol(j)*(a_minor(i)-rval(j))/(delta_ITB_ne)
        End If
                A3_ne = A3_ne + diff_vol(j)
End Do
ne_9(i) = ne(i)*A3_ne/(A1_ne+A2_ne)
ne_0(i) = ne_9(i)*(gas_ne_C1*gasa_cal(i)+gas_ne_C2+1.0)
!Time profiles of radial density information
Do j = 1,rad_pts
        If (rval(j).lt.(a_minor(i)-delta_ITB_ne)) Then
                ne_rad(j) = (ne_0(i) - ne_9(i)) * (1.0 - rval(j)**2.0/(a_minor(i) - delta_ITB_ne)**2)**nu_ne(i) + ne_9(i)
                nz3_rad(j) = ne_rad(j) / (35.0)
                ni_rad(j) = ne_rad(j) - 6.0*nz3_rad(j)
        Else
                ne_rad(j) = ne_9(i) *(a_minor(i) - rval(j)) / delta_ITB_ne
```

```
                nz3_rad(j) = ne_rad(j) / (35.0)
                ni_rad(j) = ne_rad(j) - 6.0*nz3_rad(j)
        End If
End Do
If (1.eq.0) Then
        Print *,''
        Print *,'i j ne_rad(j) ne_0(i) ne_9(i) rval(j) a_minor(i) delta_ITB_ne nu_ne(i)'
        Do j = 1,rad_pts
                Print *,i, j ,ne_rad(j),ne_0(i),ne_9(i),rval(j),a_minor(i),delta_ITB_ne,nu_ne(i),
(ne_0(i) - ne_9(i)),(1.0 - rval(j)**2.0/(a_minor(i) - delta_ITB_ne)**2)**nu_ne(i)
        End Do
        Print *,''
End If



!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! ION TEMPERATURE
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Time profiles of Ti_0 and Ti_9. Hold nu_Ti constant so Ti_5 not necessary
A1_Ti = 0
A2_Ti = 0
A3_Ti = 0
Do j=1,rad_pts-1
        If (rval(j).lt.(a_minor(i)-delta_ITB_Ti)) Then
                A1_Ti = A1_Ti + ni_rad(j)*diff_vol(j)*((gas_Ti_C1*gasa_cal(i)+gas_Ti_C2)*(1.0-(rval
(j)/(a_minor(i)-delta_ITB_Ti))**2.0)**nu_Ti(i) + 1.0)
                !A1_Ti = A1_Ti + diff_vol(j)*(((Ti_0-Ti_9)/Ti_9)*(1.0-(rval(j)/(a_minor(i)-
delta_ITB_Ti))**2.0)**nu_Ti + 1.0)
        Else
                A2_Ti = A2_Ti + ni_rad(j)*diff_vol(j)*(a_minor(i)-rval(j))/(delta_ITB_Ti)
        End If
                A3_Ti = A3_Ti + ni_rad(j)*diff_vol(j)
End Do
Ti_9(i) = Ti_kev(i)*A3_Ti/(A1_Ti+A2_Ti)
Ti_0(i) = Ti_9(i)*(gas_Ti_C1*gasa_cal(i)+gas_Ti_C2+1.0)


!Time profiles of radial Ion Temperature information. This is done whether Ti_seg_mode = 0 or 1
Do j = 1,rad_pts
        If (rval(j).lt.(a_minor(i)-delta_ITB_Ti)) Then
                Ti_rad(j) = ((Ti_0(i) - Ti_9(i)) * (1.0 - rval(j)**2.0/(a_minor(i) -
delta_ITB_Ti)**2)**nu_Ti(i) + Ti_9(i))
                Ti_J_rad(j)   = Ti_rad(j) * e * 1000.0
        Else
                Ti_rad(j) = Ti_9(i) *(a_minor(i) - rval(j)) / delta_ITB_Ti
                Ti_J_rad(j)   = Ti_rad(j) * e * 1000.0
        End If
End Do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! ELECTRON TEMPERATURE
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!Time profiles of Te_0 and Te_9. Hold nu_Te constant so Te_5 not necessary
A1_Te = 0
A2_Te = 0
A3_Te = 0
Do j=1,rad_pts-1
        If (rval(j).lt.(a_minor(i)-delta_ITB_Te)) Then
                A1_Te = A1_Te + diff_vol(j)*((gas_Te_C1*gasa_cal(i)+gas_Te_C2)*(1.0-(rval(j)/
(a_minor(i)-delta_ITB_Te))**2.0)**nu_Te(i) + 1.0)
                !A1_Te = A1_Te + diff_vol(j)*(((Te_0-Te_9)/Te_9)*(1.0-(rval(j)/(a_minor(i)-
delta_ITB_Te))**2.0)**nu_Te + 1.0)
        Else
                A2_Te = A2_Te + diff_vol(j)*(a_minor(i)-rval(j))/(delta_ITB_Te)
        End If
                A3_Te = A3_Te + diff_vol(j)
End Do
Te_9(i) = Te_keV(i)*A3_Te/(A1_Te+A2_Te)
```

```
Te_0(i) = Te_9(i)*(gas_Te_C1*gasa_cal(i)+gas_Te_C2+1.0)

!Time profiles of radial electron temperature information.
Do j = 1,rad_pts
        If (rval(j).lt.(a_minor(i)-delta_ITB_Te)) Then
                Te_rad(j) = ((Te_0(i) - Te_9(i)) * (1.0 - rval(j)**2.0/(a_minor(i) - delta_ITB_Te)**2)**nu_Te(i) + Te_9(i))
                Te_J_rad(j)   = Te_rad(j) * e * 1000.0
        Else
                Te_rad(j) = Te_9(i) *(a_minor(i) - rval(j)) / delta_ITB_Te
                Te_J_rad(j)   = Te_rad(j) * e * 1000.0
        End If
End Do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Q_ie calculation
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Do j = 1,rad_pts
                coul_log_rad(j) = log(12.0*pi*sqrt((8.854*Ti_rad(j)/1.6021)**3.0*1.0E11/ni_rad(j)))
                Q_ie_rad(j) = (ne_rad(j)*ni_rad(j))*(1.6021**4.0)*(m_e_kg_reduced/m_D_kg_reduced)*coul_log_rad(j)*(Te_J_rad(j)-Ti_J_rad(j)) / (((2*pi)**(1.5))*(8.854**2.0)*sqrt(m_e_kg)*((Te_J_rad(j))**1.5)) * 1E-14 / 1E6
                Q_ie_rad_vol(j) = Q_ie_rad(j) * diff_vol(j)
End Do
Q_ie_rad_vol(rad_pts) = 0 !was getting an NaN at rho=1 for Q_ie_rad. It's zero because the temperatures are the same. Forcing to zero.

!DENSV2_CALC(i) = dot_product(ne_rad,chrd_wt)
If (Q_ie_mult.eq.0) THen
        Q_ie(i) = 0
Else
        Q_ie(i) = Q_ie_mult*sum(Q_ie_rad_vol)/sum(diff_vol)
End If

If(isnan(Q_ie(i))) Q_ie(i) = 0.0



End Subroutine
```

```
Subroutine Simulation
Use Variables
Implicit NONE
RMP_status = 0

Open (unit = 10,FILE = outfile8)
If (sim_mode.eq.4) Then
        Write (10,*) 't_exp H_i_energy H_e_energy H_i_part ni_19 ne_19 nz1_19 nz2_19 nz3_19 nz4_19 nz5_19 nz6_19 nz7_19 nz8_19 nz9_19 nz10_19 Ti_keV Te_keV nu_ne nu_Ti nu_Te'
End If


!Find i for when to start delayed source from recycling
!Do i=1,array_size1
!       If(t_sim(i).ge.0.050) Then
!               delay_si1 = i
!               Exit
!       End If
!End Do
!Do i=1,array_size1
!       If(t_sim(i).ge.0.075) Then
!               delay_si2 = i
!               Exit
!       End If
!End Do
!Do i=1,array_size1
!       If(t_sim(i).ge.0.100) Then
!               delay_si3 = i
!               Exit
!       End If
!End Do
!Do i=1,array_size1
!       If(t_sim(i).ge.0.125) Then
!               delay_si4 = i
!               Exit
!       End If
!End Do
!Do i=1,array_size1
!       If(t_sim(i).ge.0.150) Then
!               delay_si5 = i
!               Exit
!       End If
!End Do
!Do i=1,array_size1
!       If(t_sim(i).ge.0.175) Then
!               delay_si6 = i
!               Exit
!       End If
!End Do
!Do i=1,array_size1
!       If(t_sim(i).ge.0.200) Then
!               delay_si7 = i
!               Exit
!       End If
!End Do
!Do i=1,array_size1
!       If(t_sim(i).ge.0.225) Then
!               delay_si8 = i
!               Exit
!       End If
!End Do
!Do i=1,array_size1
!       If(t_sim(i).ge.0.250) Then
!               delay_si9 = i
!               Exit
!       End If
!End Do
!Do i=1,array_size1
!       If(t_sim(i).ge.0.275) Then
!               delay_si10 = i
```

```
!               Exit
!       End If
!End Do

h_update_count = 0.0

!START MAIN DO LOOP FOR SHOT
fi_count = 0
i=1
ni_h_update_count = 0

Do
        !CALCULATE ION AND ELECTRON ENERGY DEPOSITION RATIOS
        !Calculate ion and electron energy deposition ratios
        !If (t_sim(i).ge.100*fi_count) Then

        Call fast_ion

        !       fi_count = fi_count + 1
        !Else
        !       nbi_e_frac(i) = nbi_e_frac(i-1)
        !       nbi_i_frac(i) = nbi_i_frac(i-1)
        !End If

        !CALCULATE FUSION REACTIVITY
        t_mode = 1 !tritium mode
        If(t_mode.eq.1) Then !t_mode=1 --> D-D plasma
                C1 = 5.65718E-12
                C2 = 3.41267E-3
                C3 = 1.99167E-3
                C4 = 0
                C5 = 1.05060E-5
                C6 = 0
                C7 = 0
                B_g = 31.3970
                m_bh = 937814
        Else If(t_mode.eq.2) Then !t_mode=2 --> 50/50 D-T plasma
                C1 = 1.17302E-9
                C2 = 1.51361E-2
                C3 = 7.51886E-2
                C4 = 4.60643E-3
                C5 = 1.35000E-2
                C6 = -1.06750E-4
                C7 = 1.366E-5
                B_g = 34.3827
                m_bh = 1124656
        End If

        theta_bh = Ti_keV(i)/(1-(Ti_keV(i)*(C2+Ti_keV(i)*(C4+Ti_keV(i)*C6)))/(1+Ti_keV(i)*(C3+Ti_keV
    (i)*(C5+Ti_keV(i)*C7))))
        squiggle_bh = ((B_g**2.0)/(4.0*theta_bh))**(1.0/3.0)
        sig_v(i) = C1*theta_bh*sqrt(squiggle_bh/(m_bh*Ti_kev(i)**3))*exp(-3.0*squiggle_bh) ! in cm3/
    s
        sig_v(i) = sig_v(i) * 1E-6 ! in m3/s

        !coul_log(i) = log(12.0*pi*sqrt((8.854*Ti_keV(i)/1.6021)**3.0*1.0E11/ni(i)))

        !GENERATE NEW RADIAL QUANTITIES AND CALCULATE Q_ie IF IN MODE 1
        If (sim_mode.eq.1.and.Q_ie_mult.ne.0) Call radial

        !CALCULATE ITER-98 CONFINEMENT TIME
        tau_e(i) =      0.0562 * &
                        IP(i)**0.93 * &
                        BT0_ABS(i)**0.15 * &
                        (ne(i)/10)**0.41 * &
                        m_d**0.19 * &
                        R0(i)**1.97 * &
                        ELONG_A(i)**0.78 * &
                        (R0(i)/a_minor(i))**(0.58) / &
                        PTOT_raw(i)**0.69
```

```
138
139           !CALCULATE POWER LOSS FROM RADIATION
140           PR_IMP1(i)  = (1.0 + 0.3*Te_keV(i)) * 10.0**(-43.0) * ne_19(i) * nz1_19(i) * z(1)**(3.7 -
      0.33 * log(Te_keV(i)))
141           PR_IMP2(i)  = (1.0 + 0.3*Te_keV(i)) * 10.0**(-43.0) * ne_19(i) * nz2_19(i) * z(2)**(3.7 -
      0.33 * log(Te_keV(i)))
142           PR_IMP3(i)  = (1.0 + 0.3*Te_keV(i)) * 10.0**(-43.0) * ne_19(i) * nz3_19(i) * z(3)**(3.7 -
      0.33 * log(Te_keV(i)))
143           PR_IMP4(i)  = (1.0 + 0.3*Te_keV(i)) * 10.0**(-43.0) * ne_19(i) * nz4_19(i) * z(4)**(3.7 -
      0.33 * log(Te_keV(i)))
144           PR_IMP5(i)  = (1.0 + 0.3*Te_keV(i)) * 10.0**(-43.0) * ne_19(i) * nz5_19(i) * z(5)**(3.7 -
      0.33 * log(Te_keV(i)))
145           PR_IMP6(i)  = (1.0 + 0.3*Te_keV(i)) * 10.0**(-43.0) * ne_19(i) * nz6_19(i) * z(6)**(3.7 -
      0.33 * log(Te_keV(i)))
146           PR_IMP7(i)  = (1.0 + 0.3*Te_keV(i)) * 10.0**(-43.0) * ne_19(i) * nz7_19(i) * z(7)**(3.7 -
      0.33 * log(Te_keV(i)))
147           PR_IMP8(i)  = (1.0 + 0.3*Te_keV(i)) * 10.0**(-43.0) * ne_19(i) * nz8_19(i) * z(8)**(3.7 -
      0.33 * log(Te_keV(i)))
148           PR_IMP9(i)  = (1.0 + 0.3*Te_keV(i)) * 10.0**(-43.0) * ne_19(i) * nz9_19(i) * z(9)**(3.7 -
      0.33 * log(Te_keV(i)))
149           PR_IMP10(i) = (1.0 + 0.3*Te_keV(i)) * 10.0**(-43.0) * ne_19(i) * nz10_19(i) * z(10)**(3.7 -
      0.33 * log(Te_keV(i)))
150
151           PR_IMP_TOT(i) = PR_IMP1(i)+PR_IMP2(i)+PR_IMP3(i)+PR_IMP4(i)+PR_IMP5(i)+PR_IMP6(i)+PR_IMP7(i)
      +PR_IMP8(i)+PR_IMP9(i)+PR_IMP10(i) !units of MW/m3
152           PR_BREM(i) = 1.7*10**(-38.0) * Z_eff(i) * ni(i) * ne(i) * sqrt(Te_keV(i)) / (10.0**6.0) !
      units of MW/m3
153           PRAD(i) = PR_IMP_TOT(i) + PR_BREM(i)
154
155           ! Calculate delayed source from recycling. I'm using 10 delayed groups because using only
      one group was a little too choppy
156           ! in the time trace of the calculated particle confinement time.
157   !       If (del_source.eq.1) Then
158   !               recycle_factor = 0.75    !What percentage of particles that leave the plasma
      ultimately that are not reflected that ultimately come back.
159
160   !               If (i.ge.delay_si1) Then
161   !                       S_i_del_vol1(i) = 0.1*recycle_factor*(1.0-S_ref_factor)*(ni_19(i-delay_si1
      +1)/(H_i_part(i-delay_si1+1)*tau_e(i-delay_si1+1)))
162   !               Else
163   !                       S_i_del_vol1(i) = 0
164   !               End If
165   !               If (i.ge.delay_si2) Then
166   !                       S_i_del_vol2(i) = 0.1*recycle_factor*(1.0-S_ref_factor)*(ni_19(i-delay_si2
      +1)/(H_i_part(i-delay_si2+1)*tau_e(i-delay_si2+1)))
167   !               Else
168   !                       S_i_del_vol2(i) = 0
169   !               End If
170   !               If (i.ge.delay_si3) Then
171   !                       S_i_del_vol3(i) = 0.1*recycle_factor*(1.0-S_ref_factor)*(ni_19(i-delay_si3
      +1)/(H_i_part(i-delay_si3+1)*tau_e(i-delay_si3+1)))
172   !               Else
173   !                       S_i_del_vol3(i) = 0
174   !               End If
175   !               If (i.ge.delay_si4) Then
176   !                       S_i_del_vol4(i) = 0.1*recycle_factor*(1.0-S_ref_factor)*(ni_19(i-delay_si4
      +1)/(H_i_part(i-delay_si4+1)*tau_e(i-delay_si4+1)))
177   !               Else
178   !                       S_i_del_vol4(i) = 0
179   !               End If
180   !               If (i.ge.delay_si5) Then
181   !                       S_i_del_vol5(i) = 0.1*recycle_factor*(1.0-S_ref_factor)*(ni_19(i-delay_si5
      +1)/(H_i_part(i-delay_si5+1)*tau_e(i-delay_si5+1)))
182   !               Else
183   !                       S_i_del_vol5(i) = 0
184   !               End If
185   !               If (i.ge.delay_si6) Then
186   !                       S_i_del_vol6(i) = 0
187   !               End If
188   !               If (i.ge.delay_si7) Then
```

```
!                       S_i_del_vol7(i) = 0.1*recycle_factor*(1.0-S_ref_factor)*(ni_19(i-delay_si7+1)/(H_i_part(i-delay_si7+1)*tau_e(i-delay_si7+1)))
!               Else
!                       S_i_del_vol7(i) = 0
!               End If
!               If (i.ge.delay_si8) Then
!                       S_i_del_vol8(i) = 0.1*recycle_factor*(1.0-S_ref_factor)*(ni_19(i-delay_si8+1)/(H_i_part(i-delay_si8+1)*tau_e(i-delay_si8+1)))
!               Else
!                       S_i_del_vol8(i) = 0
!               End If
!               If (i.ge.delay_si9) Then
!                       S_i_del_vol9(i) = 0.1*recycle_factor*(1.0-S_ref_factor)*(ni_19(i-delay_si9+1)/(H_i_part(i-delay_si9+1)*tau_e(i-delay_si9+1)))
!               Else
!                       S_i_del_vol9(i) = 0
!               End If
!               If (i.ge.delay_si10) Then
!                       S_i_del_vol10(i) = 0.1*recycle_factor*(1.0-S_ref_factor)*(ni_19(i-delay_si10+1)/(H_i_part(i-delay_si10+1)*tau_e(i-delay_si10+1)))
!               Else
!                       S_i_del_vol10(i) = 0
!               End If
!
!               S_i_del_vol_tot(i) =    S_i_del_vol1(i) + S_i_del_vol2(i) + S_i_del_vol3(i) + S_i_del_vol4(i) + S_i_del_vol5(i) + &
!                                       S_i_del_vol6(i) + S_i_del_vol7(i) + S_i_del_vol8(i) + S_i_del_vol9(i) + S_i_del_vol10(i)
!
!               If (S_i_del_vol_tot(i).le.1.5E20) S_i_del_vol_tot(i) = 1.5E20
!
!       Else if (del_source.eq.0) Then
!               S_i_del_vol_tot(i) = 0
!               !S_i_del_vol_tot(i) = 1.0E20
!       End If

        !DETERMINE THE RMP STATUS FOR EACH TIME STEP
        If (t_exp(i).ge.RMP_start.and.t_exp(i).lt.RMP_end) Then
                RMP_status=1
        Else
                RMP_status=0
        End If

        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !Particle and Power balances
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        !If (i.le.20) Then
        !       Print *,        PNBI(i)/volume(i), Paux_e(i)/volume(i), ECH_time_on(i), Paux_i(i)/volume(i), &
        !                       ICH_time_on(i), q95_raw(i), IP(i)/xsec_area(i), BT0_abs(i), B_dir(i), &
        !                       S_i_gas(i)/volume(i), tritop_raw(i), tribot_raw(i), nbi_i_frac(i), Ti_kev(i)/Te_kev(i), &
        !                       nu_Ti(i), nu_Te(i), nu_ne(i)

        If (sim_mode.eq.1) Then

                !CALCULATE H-FACTORS
                If (h_reuse.eq.0) Then
                        If (t_sim(i).ge.ni_h_update_count*1000) Then
                                ni_h_update = ni(i)
                                ni_h_update_count = ni_h_update_count + 1
                        End If

                        !pauxeon = 0
                        !pauxion = 0
                        !counterfracon = 0
                        !shortfracall = 1
                        !gason = 0
```

```
250
251                         !if (Paux_e_vol(i).gt.0) pauxeon = 1
252                         !if (Paux_i_vol(i).gt.0) pauxion = 1
253                         !if (nbi_counter_frac(i).gt.0) counterfracon = 1
254                         !if (nbi_short_frac(i).lt.1) shortfracall = 0
255                         !if (S_i_gas(i)/volume(i).ge.5.0E17) gason = 1
256
257                         H_i_energy(i) = Hei1 * 1.0 + &
258                                         Hei2 * PNBI(i)/volume(i) + &
259                                         Hei3 * nbi_counter_frac(i) + &
260                                         Hei4 * nbi_short_frac(i) + &
261                                         Hei5 * Paux_e_vol(i) + &
262                                         Hei6 * Paux_i_vol(i) + &
263                                         Hei7 * q95_raw(i) + &
264                                         Hei8 * q0_raw(i) + &
265                                         Hei9 * IP(i)/xsec_area(i) + &
266                                         Hei10 * BT0_abs(i) + &
267                                         Hei11 * S_i_gas(i)/volume(i)*1E-19 + &
268                                         Hei12 * tri_divert(i) + &
269                                         Hei13 * tri_nondivert(i) + &
270                                         Hei14 * ELONG_A(i) + &
271                                         Hei15 * 0.0 + &
272                                         Hei16 * 0.0 + &
273                                         Hei17 * 0.0 + &
274                                         Hei18 * 0.0 + &
275                                         Hei19 * 0.0 + &
276                                         Hei20 * 0.0
277                         !H_i_energy(i) = 1.0
278
279                         H_e_energy(i) = Hee1 * 1.0 + &
280                                         Hee2 * PNBI(i)/volume(i) + &
281                                         Hee3 * nbi_counter_frac(i) + &
282                                         Hee4 * nbi_short_frac(i) + &
283                                         Hee5 * Paux_e_vol(i) + &
284                                         Hee6 * Paux_i_vol(i) + &
285                                         Hee7 * q95_raw(i) + &
286                                         Hee8 * q0_raw(i) + &
287                                         Hee9 * IP(i)/xsec_area(i) + &
288                                         Hee10 * BT0_abs(i) + &
289                                         Hee11 * S_i_gas(i)/volume(i)*1E-19 + &
290                                         Hee12 * tri_divert(i) + &
291                                         Hee13 * tri_nondivert(i) + &
292                                         Hee14 * ELONG_A(i) + &
293                                         Hee15 * 0.0 + &
294                                         Hee16 * 0.0 + &
295                                         Hee17 * 0.0 + &
296                                         Hee18 * 0.0 + &
297                                         Hee19 * 0.0 + &
298                                         Hee20 * 0.0
299                         !H_e_energy(i) = 1.0
300
301                         H_i_part(i) = Hpi1 * 1.0 + &
302                                       Hpi2 * PNBI(i)/volume(i) + &
303                                       Hpi3 * nbi_counter_frac(i) + &
304                                       Hpi4 * nbi_short_frac(i) + &
305                                       Hpi5 * Paux_e_vol(i) + &
306                                       Hpi6 * Paux_i_vol(i) + &
307                                       Hpi7 * q95_raw(i) + &
308                                       Hpi8 * q0_raw(i) + &
309                                       Hpi9 * IP(i)/xsec_area(i) + &
310                                       Hpi10 * BT0_abs(i) + &
311                                       Hpi11 * S_i_gas(i)/volume(i)*1E-19 + &
312                                       Hpi12 * tri_divert(i) + &
313                                       Hpi13 * tri_nondivert(i) + &
314                                       Hpi14 * ELONG_A(i) + &
315                                       Hpi15 * 0.0 + &
316                                       Hpi16 * 0.0 + &
317                                       Hpi17 * 0.0 + &
318                                       Hpi18 * 0.0 + &
319                                       Hpi19 * 0.0 + &
```

```
                                        Hpi20 * 0.0
                        H_i_part(i) = 1.0

                Else If (h_reuse.eq.1) Then
                        !do nothing. the h and nu values have already been read in in the timeseries subroutine.
                        ! We do still have to calculate S_i, which has to be done after the H-factors are either calculated or read in.
                End If

                !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                !Ion particle balance terms
                !S_i_del_vol_tot(i) = S_reem_factor*ni_19(i) / ((1.0) * tau_e(i))
                ! H_i_part = 1 is used for the re-emitted source. Using the regression model H_i_part at each time step introduces problems
                ! and the result is much worse than if H_i_part is left at 1. The reflected source is still based on "actual" confinement losses
                ! leaving the plasma.
                S_i_rec_vol_tot(i) =  S_rec_factor*ni_19(i) / (H_i_part(i) * tau_e(i))
                S_i(i) = S_i_ext_vol(i) + S_i_rec_vol_tot(i) !external + recycled
                fus_loss_i(i) = 0.5*ni_19(i)*sig_v(i)*ni_19(i)
                conf_p(i) =ni_19(i)/(H_i_part(i)*tau_e(i))

                !Electron power balance terms
                P_ohm(i) = POH_vol(i)
                P_e(i) = Paux_e_vol(i) + (PNBI(i)*nbi_net_frac(i)*nbi_e_frac(i))/volume(i) !ECH + beams
                Pfus_e(i) = 0.25*ni_19(i)*sig_v(i)*ni_19(i)*(U_alpha_DT_MJ*alpha_e_frac)
                Coll(i) = Q_ie(i)
                Rad(i) = PRAD(i)
                conf_e(i) =1.5*ne_19(i)*Te_MJ(i)/(H_e_energy(i)*tau_e(i))

                !Ion power balance terms
                P_i(i) = Paux_i_vol(i) + (PNBI(i)*nbi_net_frac(i)*nbi_i_frac(i))/volume(i) !ICH + beams
                Pfus_i(i) = 0.25*ni_19(i)*sig_v(i)*ni_19(i)*(U_alpha_DT_MJ*alpha_i_frac)
                conf_i(i) =1.5*ni_19(i)*Ti_MJ(i)/(H_i_energy(i)*tau_e(i))
                !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

                i_max = i
                i=i+1

                If (ni_sim.eq.1) Then
                        ni_19(i) = ni_19(i-1) + delta_t*(S_i(i-1) - fus_loss_i(i-1) - conf_p(i-1))

                        nz1_19(i) = nz1_19(i-1)
                        nz2_19(i) = nz2_19(i-1)
                        nz3_19(i) = nz3_19(i-1)
                        nz4_19(i) = nz4_19(i-1)
                        nz5_19(i) = nz5_19(i-1)
                        nz6_19(i) = nz6_19(i-1)
                        nz7_19(i) = nz7_19(i-1)
                        nz8_19(i) = nz8_19(i-1)
                        nz9_19(i) = nz9_19(i-1)
                        nz10_19(i) = nz10_19(i-1)

                        ne_19(i) = ni_19(i) + nz1_19(i)*z(1) + nz2_19(i)*z(2) + nz3_19(i)*z(3) + nz4_19(i)*z(4) + nz5_19(i)*z(5) + &
                                                    nz6_19(i)*z(6) + nz7_19(i)*z(7) + nz8_19(i)*z(8) + nz9_19(i)*z(9) + nz10_19(i)*z(10)

                End If

                If (Te_sim.eq.1) Then
                        Te_MJ(i) = Te_MJ(i-1) + delta_t/ne_19(i-1)*(2.0/3.0*(P_ohm(i-1) + P_e(i-1) + Pfus_e(i-1) - Coll(i-1) - rad(i-1) - conf_e(i-1))-Te_MJ(i-1)*ne_19_td(i-1))
                End If

```

```
                If (Ti_sim.eq.1) Then
                        Ti_MJ(i) = Ti_MJ(i-1) + delta_t/ni_19(i-1)*(2.0/3.0*(             P_i(i-1) + Pfus_i(i-1) + Coll(i-1)            - conf_i(i-1))-Ti_MJ(i-1)*ni_19_td(i-1))
                End If
        Else if (sim_mode.eq.4) Then

                !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                !Ion particle balance terms
                !S_i_del_vol_tot(i) = S_reem_factor*ni_19(i)/(H_i_part(i)*tau_e(i))
                S_i_rec_vol_tot(i) = S_rec_factor*ni_19(i)/(H_i_part(i)*tau_e(i))
                S_i(i) = S_i_ext_vol(i) + S_i_rec_vol_tot(i) !external + recycled
                fus_loss_i(i) = 0.5*ni_19(i)*sig_v(i)*ni_19(i)
                conf_p(i) =ni_19(i)/(H_i_part(i)*tau_e(i))

                !Electron power balance terms
                P_ohm(i) = POH_vol(i)
                P_e(i) = Paux_e_vol(i) + (PNBI(i)*nbi_net_frac(i)*nbi_e_frac(i))/volume(i) !ECH + beams
                Pfus_e(i) = 0.25*ni_19(i)*sig_v(i)*ni_19(i)*(U_alpha_DT_MJ*alpha_e_frac)
                Coll(i) = Q_ie(i)
                Rad(i) = PRAD(i)
                conf_e(i) =1.5*ne_19(i)*Te_MJ(i)/(H_e_energy(i)*tau_e(i))

                !Ion power balance terms
                P_i(i) = Paux_i_vol(i) + (PNBI(i)*nbi_net_frac(i)*nbi_i_frac(i))/volume(i) !ICH + beams
                Pfus_i(i) = 0.25*ni_19(i)*sig_v(i)*ni_19(i)*(U_alpha_DT_MJ*alpha_i_frac)
                conf_i(i) =1.5*ni_19(i)*Ti_MJ(i)/(H_i_energy(i)*tau_e(i))
                !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

                If (H_i_part_1.eq.1) Then
                        H_i_part(i) = 1.0
                Else
                        H_i_part(i) = Hpi1 * 1.0 + &
                                        Hpi2 * PNBI(i)/volume(i) + &
                                        Hpi3 * nbi_counter_frac(i) + &
                                        Hpi4 * nbi_short_frac(i) + &
                                        Hpi5 * Paux_e_vol(i) + &
                                        Hpi6 * Paux_i_vol(i) + &
                                        Hpi7 * q95_raw(i) + &
                                        Hpi8 * q0_raw(i) + &
                                        Hpi9 * IP(i)/xsec_area(i) + &
                                        Hpi10 * BT0_abs(i) + &
                                        Hpi11 * S_i_gas(i)/volume(i)*1E-19 + &
                                        Hpi12 * tri_divert(i) + &
                                        Hpi13 * tri_nondivert(i) + &
                                        Hpi14 * ELONG_A(i) + &
                                        Hpi15 * 0.0 + &
                                        Hpi16 * 0.0 + &
                                        Hpi17 * 0.0 + &
                                        Hpi18 * 0.0 + &
                                        Hpi19 * 0.0 + &
                                        Hpi20 * 0.0
                End If

                If (S_i_rec_fit.eq.1) Then
                        S_i_rec_vol_tot(i) = ni_19_td(i) - S_i_ext_vol(i) + fus_loss_i(i) + ni_19(i)/(H_i_part(i)*tau_e(i))
                Else
                        H_i_part(i) = ni_19(i)*(1.0 - S_rec_factor)/(tau_e(i)*(S_i_ext_vol(i) - fus_loss_i(i) - ni_19_td(i)))
                End If

                H_e_part(i) = H_i_part(i)
                H_e_energy(i) = 1.5*ne_19(i)*Te_MJ(i)/(tau_e(i)*(-1.5*(ne_19(i)*Te_MJ_td(i) + Te_MJ(i)*ne_19_td(i)) + P_ohm(i) + P_e(i) + Pfus_e(i) - Coll(i) - rad(i)))
                H_i_energy(i) = 1.5*ni_19(i)*Ti_MJ(i)/(tau_e(i)*(-1.5*(ni_19(i)*Ti_MJ_td(i) + Ti_MJ(i)*ni_19_td(i))            + P_i(i) + Pfus_i(i) + Coll(i)         ))
```

```
444
445                 If (isnan(H_e_energy(i))) H_e_energy(i) = 0.0
446                 If (isnan(H_i_energy(i))) H_i_energy(i) = 0.0
447                 If (isnan(H_i_part(i))) H_i_part(i) = 0.0
448
449                 i_max = i
450                 i=i+1
451
452         End If
453
454         !UNIT CONVERSIONS
455         ni(i) = ni_19(i)*1E-19
456
457         nz1(i) = nz1_19(i)*1E-19
458         nz2(i) = nz2_19(i)*1E-19
459         nz3(i) = nz3_19(i)*1E-19
460         nz4(i) = nz4_19(i)*1E-19
461         nz5(i) = nz5_19(i)*1E-19
462         nz6(i) = nz6_19(i)*1E-19
463         nz7(i) = nz7_19(i)*1E-19
464         nz8(i) = nz8_19(i)*1E-19
465         nz9(i) = nz9_19(i)*1E-19
466         nz10(i) = nz10_19(i)*1E-19
467
468         ne(i) = ne_19(i)*1E-19
469
470         Te_J(i) = Te_MJ(i) * 1.0E6
471         Te_keV(i) = Te_MJ(i)*1000/e
472
473         Ti_J(i) = Ti_MJ(i) * 1.0E6
474         Ti_keV(i) = Ti_MJ(i)*1000/e
475
476         Ti_Te(i) = Ti_keV(i)/Te_kev(i)
477
478         If (sim_mode.eq.4.and.t_exp(i-1).le.(end_time-0.05)) Then
479                 Write (10,*) t_exp(i-1), H_i_energy(i-1), H_e_energy(i-1), H_i_part(i-1), ni_19
    (i-1), ne_19(i-1), nz1_19(i-1), nz2_19(i-1), nz3_19(i-1), nz4_19(i-1), nz5_19(i-1), nz6_19(i-1),
    nz7_19(i-1),nz8_19(i-1), nz9_19(i-1), nz10_19(i-1), Ti_keV(i-1), Te_keV(i-1), nu_ne(i-1), nu_Ti
    (i-1), nu_Te(i-1)
480         End If
481
482         actual_end_time = t_exp(i)
483         If(sim_mode.eq.1) Then
484                 !If (i.ge.24630) Then
485                 If (isnan(ni_19(i)).or.isnan(Te_MJ(i)).or.isnan(Ti_MJ(i))) Then
486                         Print *,''
487                         Print *,'i = ',i
488                         Print *,'t_exp(i) = ',t_exp(i)
489                         Print *,'ne(i) = ',ne(i)
490                         Print *,'ni(i) = ',ni(i)
491                         Print *,'Te_keV(i) = ',Te_keV(i)
492                         Print *,'Ti_keV(i) = ',Ti_keV(i)
493                         Print *,'diff_ni_t(i-1) = ',diff_ni_t(i-1)
494                         Print *,'diff_Ti_t(i-1) = ',diff_Ti_t(i-1)
495                         Print *,'diff_Te_t(i-1) = ',diff_Te_t(i-1)
496                         Print *,'diff_neTe_t(i-1) = ',diff_neTe_t(i-1)
497                         Print *,'diff_niTi_t(i-1) = ',diff_niTi_t(i-1)
498                         Print *,'POH_vol(i-1) = ',POH_vol(i-1)
499                         Print *,'P_nbi_e_vol(i-1) = ',P_nbi_e_vol(i-1)
500                         Print *,'Paux_e_vol(i-1) = ',Paux_e_vol(i-1)
501                         Print *,'P_alpha_e(i-1) = ',P_alpha_e(i-1)
502                         Print *,'Q_ie(i-1) = ',Q_ie(i-1)
503                         Print *,'PRAD(i-1) = ',PRAD(i-1)
504                         Print *,'tau_e_energy(i-1) = ',tau_e_energy(i-1)
505                         Print *,''
506                         Print *,'j ne_rad Te_rad Ti_rad coul_log_rad Q_ie_rad Q_ie_rad_vol'
507
508                         Do j = 1,rad_pts
509                                 Print *,j ,ne_rad(j),Te_rad(j),Ti_rad(j),coul_log_rad(j),Q_ie_rad
    (j),Q_ie_rad_vol(j)
```

```
510                         End Do
511                         Print *,''
512
513                         If (isnan(ni_19(i)).or.isnan(Te_MJ(i)).or.isnan(Ti_MJ(i))) Exit !this
    condition is duplicated for when I'm printing out all i ge some number
514                   End If
515         End If
516
517         If(mod(i,100).eq.0) Print *,'t_exp = ',t_exp(i)
518
519         !Set output_mode to zero if you're only interested in doing one timestep to get the
    calculated H-factors
520         !This can be useful when looking at shots in equilibrium when you only need the equilibrium
    confinement parameters
521
522         If(output_mode.eq.0) exit_flag = 1
523         If(exit_flag.eq.1) Exit
524         If(t_exp(i).ge.end_time) Exit
525   End Do
526   Close (10)
527   End Subroutine
```

```fortran
Subroutine Timeseries
Use Variables
Implicit NONE
EXTERNAL DGELS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! POPULATE TIME ARRAYS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
t_sim(1) = 0
t_exp(1) = start_time
Do i=2,array_size1
        t_sim(i) = t_sim(i-1)+delta_t
        t_exp(i) = t_exp(i-1)+delta_t
End Do

inquire (file=outfile1,EXIST=file_exists)
If (file_exists) Then !If quickinput exists, then read that in.
        Print *,'Synced data file already exists. This will be much faster.'
        Open (unit = 13, File = outfile1)
        Do i=1,array_size1
                If (t_exp(i).gt.end_time) Exit
                Read (13,*) t_exp(i), (dat_synced(i,k), k=1,array_size6)
        End Do
        Close (13)
Else !If quickinput does not exist, create it using the smoothed out raw data. If that doesn't exist, create it.
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        ! READ IN EXPERIMENTAL TIMESERIES DATA
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        smooth_window = .025 !+/- each way, 0.025 seconds tends to work well

        ! Check if smoothed out data file exists
        inquire (file=datafile_smooth,EXIST=file_exists)

        ! If smoothed file exists, read in first line to verify smoothing window size. If correct, read in data
        If (file_exists) Then
                !Verify smoothing window width
                Open (unit = 20, File = datafile_smooth)
                Read (20,*) max_pts, smooth_window_check
                If (smooth_window_check.eq.smooth_window) Then
                        !Read in smoothed data
                        Print *,"smoothed data file exists and has correct smoothing window"
                        Print *,"reading smoothed data"
                        Do i = 1,max_pts
                                Read (20,*) (dat_time(i,k), dat(i,k), k=1,array_size6)
                        End Do
                Else
                        Close (20)
                        Print *,"smoothed data file exists, but has wrong smoothing window"
                        Print *,"smoothing out raw data with specified smooth_window"
                        Call data_smooth
                        Open (unit = 20, File = datafile_smooth)
                        Print *,"reading smoothed data"
                        Read (20,*) max_pts, smooth_window_check
                        Do i = 1,max_pts
                                Read (20,*) (dat_time(i,k), dat(i,k), k=1,array_size6)
                        End Do
                End If
        Else
                Print *,"smoothed data file didn't exist"
                Print *,'smoothing out raw data'
                Call data_smooth

                Open (unit = 20, File = datafile_smooth)
                Print *,"reading smoothed data"
                Read (20,*) max_pts, smooth_window_check
                Do i = 1,max_pts
                        Read (20,*) (dat_time(i,k), dat(i,k), k=1,array_size6)
                End Do
```

```
        End If
        Close (20)

        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        ! DATA SYNC - Raw data doesn't line up with timestep in the code. If they line up, use the corresponding
        !             data value. If they don't, linearly interpolate.
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        Print *,'syncing data to specified timestep'
        Do i=1,array_size1
                If (t_exp(i).gt.end_time) Exit
                !for each t_exp, find the positions of the dat_times just less than and just greater than t_exp
                loc_prev = maxloc(dat_time,dim=1,mask=dat_time.le.t_exp(i))
                loc_after = loc_prev+1
                Do k=1,array_size6
                        If (dat_time(loc_prev(k),k).eq.t_exp(i)) Then !t_exp lines up perfectly with a raw data point
                                dat_synced(i,k) = dat(loc_prev(k),k)
                                dat_synced_td(i,k) = (dat(loc_prev(k),k)-dat(loc_prev(k)-1,k))/(dat_time(loc_prev(k),k)-dat_time(loc_prev(k)-1,k))
                        Else
                                dat_synced_td(i,k) = (dat(loc_after(k),k)-dat(loc_prev(k),k))/(dat_time(loc_after(k),k)-dat_time(loc_prev(k),k))
                                dat_synced(i,k) = dat(loc_prev(k),k) + dat_synced_td(i,k) * (t_exp(i)-dat_time(loc_prev(k),k))
                        End If
                        If (isnan(dat_synced(i,k))) dat_synced(i,k)=0
                End Do
        End Do

        Print *,'Writing synced data output file'
        Open (unit = 13, File = outfile1)
        Do i=1,array_size1
                If (t_exp(i).gt.end_time) Exit
                Write (13,*) t_exp(i), (dat_synced(i,k), k=1,array_size6)
        End Do
        Close (13)
End If
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! ASSIGN SYNCED DATA TO MEANINGFUL VARIABLE NAMES
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Do i=1,array_size1
        If (t_exp(i).gt.end_time) Exit

        H98(i) = dat_synced(i,1)
        IP(i) = dat_synced(i,2)*1E-6 !now in MA
        BT0(i) = dat_synced(i,3)

        B_dir(i) = BT0(i)/abs(BT0(i)) !either 1 or -1

        GASA_CAL(i) = dat_synced(i,4)*18.27417
        PNBI(i) = dat_synced(i,5)*1E-6 !now in MW
        POH(i) = dat_synced(i,6)*1E-6 !now in MW
        zmaxis(i) = dat_synced(i,7)
        nbi1(i) = dat_synced(i,8)*1E-6 !now in MW
        nbi1e(i) = 75.0 !Filling in nbi*e manually as 80keV for now
        nbi2(i) = dat_synced(i,9)*1E-6 !now in MW
        nbi2e(i) = 80.0 !Filling in nbi*e manually as 80keV for now
        nbi3(i) = dat_synced(i,10)*1E-6 !now in MW
        nbi3e(i) = 80.0 !Filling in nbi*e manually as 80keV for now
        nbi4(i) = dat_synced(i,11)*1E-6 !now in MW
        nbi4e(i) = 80.0 !Filling in nbi*e manually as 80keV for now
        nbi5(i) = dat_synced(i,12)*1E-6 !now in MW
        nbi5e(i) = 80.0 !Filling in nbi*e manually as 80keV for now
        nbi6(i) = dat_synced(i,13)*1E-6 !now in MW
        nbi6e(i) = 80.0 !Filling in nbi*e manually as 80keV for now
        nbi7(i) = dat_synced(i,14)*1E-6 !now in MW
        nbi7e(i) = 80.0 !Filling in nbi*e manually as 80keV for now
```

```
        nbi8(i) = dat_synced(i,15)*1E-6 !now in MW
        nbi8e(i) = 80.0 !Filling in nbi*e manually as 80keV for now

        nbi_co(i) = nbi1(i) + nbi2(i) + nbi3(i) + nbi4(i) + nbi7(i) + nbi8(i)
        nbi_counter(i) = nbi5(i) + nbi6(i)
        nbi_short(i) = nbi1(i) + nbi3(i) + nbi6(i) + nbi7(i)
        nbi_long(i) =  nbi2(i) + nbi4(i) + nbi5(i) + nbi8(i)

        nbi_counter_frac(i) = nbi_counter(i)/(nbi_counter(i)+nbi_co(i))
        nbi_short_frac(i) = nbi_short(i)/(nbi_short(i)+nbi_long(i))

        densV2(i) = dat_synced(i,16)*1E-19
        R0(i) = dat_synced(i,17)
        TSTE_TAN(i) = dat_synced(i,18)
        BDOTAMPL(i) = dat_synced(i,19)
        a_minor(i) = dat_synced(i,20)
        VOLUME(i) = dat_synced(i,21)
        ELONG_A(i) = dat_synced(i,22)
        If (dat_synced(i,23).lt.0) Then
                Paux_e(i) = 0.0
                Paux_e_vol(i) = 0.0
                Else
                Paux_e(i) = dat_synced(i,23)*1E-6 !now in MW
                Paux_e_vol(i) = dat_synced(i,23)*1E-6/volume(i) !now in MW
        End If
        If (dat_synced(i,24).lt.0.or.isnan(dat_synced(i,24))) Then
                Paux_i(i) = 0.0
                Paux_i_vol(i) = 0.0
                Else
                Paux_i(i) = dat_synced(i,24)*1E-6 !now in MW
                Paux_i_vol(i) = dat_synced(i,24)*1E-6/volume(i) !now in MW
        End If

        taue_raw(i) = dat_synced(i,25)
        tauth_raw(i) = dat_synced(i,26)
        taueh98y2_raw(i) = dat_synced(i,27)
        ptot_raw(i) = dat_synced(i,28)/1E6 !now in MW
        q95_raw(i) = dat_synced(i,29)
        li_raw(i) = dat_synced(i,30)
        tritop_raw(i) = dat_synced(i,31)
        tribot_raw(i) = dat_synced(i,32)
        If (up_divertor.eq.1) Then
                tri_divert(i) = tritop_raw(i)
                tri_nondivert(i) = tribot_raw(i)
        Else
                tri_divert(i) = tribot_raw(i)
                tri_nondivert(i) = tritop_raw(i)
        End If

        q0_raw(i) = dat_synced(i,33)

        nbi_net_frac(i) = 1.0
        BT0_abs(i) = abs(BT0(i)) !Absolute value of toroidal magnetic field
        S_i_nbi(i) = PNBI(i) * 1E3 / (nbi1e(i)*e)
        S_i_gas(i) = GASA_CAL(i)*avogadro/(R_gas*273.15)/150
        S_i_ext_vol(i) = (S_i_nbi(i) + S_i_gas(i)) / volume(i) !total volumetric ion source
        poh_vol(i) = poh(i)/volume(i) !Ohmic power injection density (MW/m3)
        nbi_tot(i) = (nbi1(i)+nbi2(i)+nbi3(i)+nbi4(i)+nbi5(i)+nbi6(i)+nbi7(i)+nbi8(i))*1E-6 !units of MW
        nbi_vol(i) = nbi_tot(i)/volume(i) !Neutral Beam power injection density (MW/m3)
        xsec_area(i) = pi*(a_minor(i)**2)*ELONG_A(i)
End Do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! ECH_TIME_ON AND ICH_TIME_ON
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!Identify time when ECH turns on and track time since that event
ech_on_flag=0
Do i=1,array_size1
```

```
        !Identify ECH start time and turn flag on if ECH > .5 MW
        If (ech_on_flag .eq. 0 .and. Paux_e(i) .gt. 0.5 ) Then
                ech_start_time = t_exp(i)
                ech_on_flag = 1
        End If

        !Identify ECH end time and turn flag off if ECH < .5 MW
        If (ech_on_flag .eq. 1 .and. Paux_e(i) .lt. 0.5 ) Then
                ech_on_flag = 0
        End If

        !Set time since ECH on to zero if ECH is off and calculate time if it is on.
        If (ech_on_flag .eq. 0) Then
                ech_time_on(i) = 0.0
        Else
                ech_time_on(i) = t_exp(i) - ech_start_time
        End If
End Do

!Identify time when ICH turns on and track time since that event
ich_on_flag=0
Do i=1,array_size1
        !Identify ICH start time and turn flag on if ICH > 0.01 MW
        If (ich_on_flag .eq. 0 .and. Paux_i(i) .gt. 0.01 ) Then
                ich_start_time = t_exp(i)
                ich_on_flag = 1
        End If

        !Identify ECH end time and turn flag off if ICH < 0.01 MW
        If (ich_on_flag .eq. 1 .and. Paux_i(i) .lt. 0.01 ) Then
                ich_on_flag = 0
        End If

        !Set time since ECH on to zero if ECH is off and calculate time if it is on.
        If (ich_on_flag .eq. 0) Then
                ich_time_on(i) = 0.0
        Else
                ich_time_on(i) = t_exp(i) - ich_start_time
        End If
End Do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! IMPORT STUFF FROM A PREVIOUS MODE 4 RUN
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

If (sim_mode.eq.1.and.h_reuse.eq.1) Then
        Open (unit = 10, File = outfile8)
        Do i=1,array_size1
                Read (10,*,IOSTAT=io) H_i_energy(i), H_e_energy(i), H_i_part(i), H_e_part(i),
dummy, dummy, dummy, nu_ne(i), nu_Ti(i), nu_Te(i)
                If(t_exp(i+1).gt.end_time.or.io.lt.0) Exit
        End Do
        Close (10)
End If

If (sim_mode.eq.1.and.h_reuse.eq.0.and.ni_sim.eq.0) Then
        Open (unit = 10, File = outfile8)
        Do i=1,array_size1
                Read (10,*,IOSTAT=io) dummy, dummy, dummy, dummy, ni_19(i), ne_19(i), nz1_19(i),
nz2_19(i), nz3_19(i), nz4_19(i), nz5_19(i), nz6_19(i), nz7_19(i),nz8_19(i), nz9_19(i), nz10_19(i),
dummy, dummy, dummy, dummy, dummy
                If(t_exp(i+1).gt.end_time.or.io.lt.0) Exit
        End Do
        Close (10)
End If
Print *,'end of timeseries'
End Subroutine
```

```
1   Module Variables
2   Implicit NONE
3
4   !General and counting integers
5   integer ::                              i, i_pre, j, j_last, k, l, m, n, p, w, i_gof, IOstatus,
    RMP_status, i_end_time, fi_count, &
6                                           array_size1, array_size2, array_size3, array_size4,
    array_size5, array_size6, array_size7, array_size8, array_size9, &
7                                           j_fi, j_fi_max, i_max, w_max, out_count, filecount,
    av_count, stop_h, start_point_av, &
8                                           ne_converge_count, Te_converge_count, Ti_converge_count,
    rad_count, exit_flag, &
9                                           ne_h_flag, Te_h_flag, Ti_h_flag, h_find_count,
    start_point_td, end_point_td, array_size_td, &
10                                          td_start, td_end, segment_count, av_point_count,
    Ti_seg_mode, Te_seg_mode, ne_seg_mode, nz3_seg_mode, &
11                                          delay_si1, delay_si2, delay_si3, delay_si4, delay_si5,
    delay_si6, delay_si7, delay_si8, delay_si9, delay_si10, &
12                                          m_ne, m_ti, m_te,  ech_on_flag, ich_on_flag, gasa_av_count,
    gas_average_time, io, del_source, &
13                                          raw_output, raw_synced_output, h_reuse, l_q_region,
    event_mode, ni_sim, Ti_sim, Te_sim, ni_h_update_count
14  character(len=11), dimension (:), allocatable ::        shotnum
15
16  !Integers used in D3D_timeseries
17  integer ::                              max_pts, slope, lwork_td, loc_prev_ne, loc_prev_Te,
    loc_prev_Ti, loc_after_ne, loc_after_Te, loc_after_Ti
18
19  !Variables read in for program control
20  real ::                         start_time, end_time, delta_t, &
21                                          fz_tot, fz_z_tot, &
22                                          ne_0_old, ne_5_old, ne_9_old, Te_0_old, Te_5_old, Te_9_old,
    Ti_0_old, Ti_5_old, Ti_9_old, &
23                                          ne_0_start, ne_5_start, ne_9_start, Te_0_start, Te_5_start,
    Te_9_start, Ti_0_start, Ti_5_start, Ti_9_start, &
24                                          dummy, H_i_energy_exp, H_e_energy_exp, H_i_part_exp,
    ni_19_exp, ne_19_exp, nz1_19_exp, nz2_19_exp, nz3_19_exp, &
25                                          nz4_19_exp, nz5_19_exp, nz6_19_exp, nz7_19_exp,nz8_19_exp,
    nz9_19_exp, nz10_19_exp, Ti_keV_exp, Te_keV_exp, &
26                                          nu_ne_exp, nu_Ti_exp, nu_Te_exp, q_ie_mult, gas_av, line1,
    line2, event_time
27
28  !Variables used internally
29  real ::                         delta_ITB_ne, delta_ITB_Te, delta_ITB_Ti, &
30                                          tol, tol_rad, ne_converge, Te_converge, Ti_converge,
    ne_start, Te_start, Ti_start, ne_old, Te_old, Ti_old, &
31                                          ne_conv_check, Te_conv_check, Ti_conv_check, &
32                                          wb_therm_J, Te_j_fi, wb_0_J, delta_t_fi, Z_b, B_fi, C_fi,
    en_range, &
33                                          en_bin_width, nbi_e_frac_sum, nbi_i_frac_sum, alpha_i_frac,
    alpha_e_frac, &
34                                          S_rec_factor, S_reem_Factor, initial_av_sum, RMP_start,
    RMP_end, &
35                                          C1, C2, C3, C4, C5, C6, C7, B_g, m_bh, theta_bh,
    squiggle_bh, &
36                                          tol_eq_check, ni_diff, ne_diff, Ti_diff, Te_diff, H98_calc,
    vol_tot, rval_v2, &
37                                          delta_ne, delta_Te, delta_Ti, &
38                                          nerad_C1, nerad_C2, nerad_C3, nerad_C4, Terad_C1, Terad_C2,
    Terad_C3, Terad_C4, Tirad_C1, Tirad_C2, Tirad_C3, Tirad_C4, &
39                                          A1_ne, A2_ne, A3_ne, A1_Te, A2_Te, A3_Te, A1_Ti, A2_Ti,
    A3_Ti, &
40                                          ne_parab_ht, Te_parab_ht, Ti_parab_ht, Te_H_diff,
    Ti_H_diff, ne_H_diff, H_i_energy_old, H_e_energy_old, H_i_part_old, &
41                                          diff_ni_t_init, diff_Ti_t_init, diff_Te_t_init,
    diff_niTi_t_init, diff_neTe_t_init, &
42                                          dens_td_init, te_td_init, ti_td_init, time_around_sp,
    ne_f2e_diff, Te_f2e_diff, Ti_f2e_diff, &
43                                          seg_start, seg_end, seg_width, temp_sum, dens_sum,
    ne_bad_elements, Te_bad_elements, recycle_factor, &
```

```
44                                      Hei1, Hei2, Hei3, Hei4, Hei5, Hei6, Hei7, Hei8, Hei9,
    Hei10, &
45                                      Hei11, Hei12, Hei13, Hei14, Hei15, Hei16, Hei17, Hei18,
    Hei19, Hei20, &
46                                      Hee1, Hee2, Hee3, Hee4, Hee5, Hee6, Hee7, Hee8, Hee9,
    Hee10, &
47                                      Hee11, Hee12, Hee13, Hee14, Hee15, Hee16, Hee17, Hee18,
    Hee19, Hee20, &
48                                      Hpi1, Hpi2, Hpi3, Hpi4, Hpi5, Hpi6, Hpi7, Hpi8, Hpi9,
    Hpi10, &
49                                      Hpi11, Hpi12, Hpi13, Hpi14, Hpi15, Hpi16, Hpi17, Hpi18,
    Hpi19, Hpi20, &
50                                      h_update_count, delta_r, ech_start_time, ich_start_time, &
51                                      event_pre_start, event_post_start, event_pre_end,
    event_post_end, ni_h_update, &
52                                      gof_width, t_start_gof, t_end_gof, ni_gof_av, Ti_gof_av,
    Te_gof_av, actual_end_time, &
53                                      smooth_window, smooth_window_check, ne_fi, ni_fi, Te_fi,
    Ti_fi, nbi_i, nbi_e, vol_tot_flat
54
55
56  integer ::                          count1, en_bins_fi, shots, output_mode, H_find_mode,
    t_mode, sim_mode, &
57                                      H_find_matrix_row, H_find_matrix_column, lwmax_td, info_td,
    up_divertor, &
58                                      pauxeon, pauxion, counterfracon, shortfracall, gason,
    H_i_part_1, S_i_rec_fit
59
60  integer, dimension (:), allocatable :: work_td, start_point, loc_prev, loc_after
61
62  real, dimension (:), allocatable ::     start_value_td, end_value_td, dens_array_td, temp_array_td,
    &
63                                      av_time, av_data, dat_synced_temp, &
64                                      Ti_time, Ti_0_seg, Ti_5_seg, Ti_9_seg, Ti_seg, &
65                                      Te_time, Te_0_seg, Te_5_seg, Te_9_seg, Te_seg, &
66                                      ne_time, ne_0_seg, ne_5_seg, ne_9_seg, ne_seg, ni_seg,
    nz3_seg, &
67                                      ne_0, ne_5, ne_9, Te_0, Te_5, Te_9, Ti_0, Ti_5, Ti_9, &
68                                      skip_array,  t_exp_clean, &
69                                      cleanup_start1, cleanup_start2, cleanup_start3,
    cleanup_start4, cleanup_start5, &
70                                      cleanup_start6, cleanup_start7, cleanup_start8,
    cleanup_start9, cleanup_start10, &
71                                      cleanup_start11, cleanup_start12, cleanup_start13,
    cleanup_start14, cleanup_start15, &
72                                      cleanup_start16, cleanup_start17, cleanup_start18,
    cleanup_start19, cleanup_start20, &
73                                      cleanup_start21, cleanup_start22, cleanup_start23,
    cleanup_start24, cleanup_start25, &
74                                      cleanup_start26, cleanup_start27, cleanup_start28,
    cleanup_start29, cleanup_start30,&
75                                      cleanup_end1, cleanup_end2, cleanup_end3, cleanup_end4,
    cleanup_end5, &
76                                      cleanup_end6, cleanup_end7, cleanup_end8, cleanup_end9,
    cleanup_end10, &
77                                      cleanup_end11, cleanup_end12, cleanup_end13, cleanup_end14,
    cleanup_end15, &
78                                      cleanup_end16, cleanup_end17, cleanup_end18, cleanup_end19,
    cleanup_end20, &
79                                      cleanup_end21, cleanup_end22, cleanup_end23, cleanup_end24,
    cleanup_end25, &
80                                      cleanup_end26, cleanup_end27, cleanup_end28, cleanup_end29,
    cleanup_end30, &
81                                      skip_lt1, skip_lt2, skip_lt3, skip_lt4, skip_lt5, &
82                                      skip_lt6, skip_lt7, skip_lt8, skip_lt9, skip_lt10, &
83                                      skip_lt11, skip_lt12, skip_lt13, skip_lt14, skip_lt15, &
84                                      skip_lt16, skip_lt17, skip_lt18, skip_lt19, skip_lt20, &
85                                      skip_lt21, skip_lt22, skip_lt23, skip_lt24, skip_lt25, &
86                                      skip_lt26, skip_lt27, skip_lt28, skip_lt29, skip_lt30, &
87                                      skip_gt1, skip_gt2, skip_gt3, skip_gt4, skip_gt5, &
```

```
 88                                          skip_gt6, skip_gt7, skip_gt8, skip_gt9, skip_gt10, &
 89                                          skip_gt11, skip_gt12, skip_gt13, skip_gt14, skip_gt15, &
 90                                          skip_gt16, skip_gt17, skip_gt18, skip_gt19, skip_gt20, &
 91                                          skip_gt21, skip_gt22, skip_gt23, skip_gt24, skip_gt25, &
 92                                          skip_gt26, skip_gt27, skip_gt28, skip_gt29, skip_gt30, &
 93                                          nu_ne, nu_Te, nu_Ti, &
 94                                          rval, diff_vol, diff_vol_frac, &
 95                                          ne_rad, ni_rad, nz3_rad, Te_rad, Ti_rad, ne_width,
    nz3_width, ni_width, Te_width, Ti_width, &
 96                                          nTi, nTe, ne_V_width, nz3_V_width, ni_V_width, Te_V_width,
    Ti_V_width, vol_width, &
 97                                          nTe_V, nTi_V, ne_19_rad, ni_19_rad, Te_keV_rad, Te_J_rad,
    Te_MJ_rad, Ti_keV_rad, &
 98                                          Ti_J_rad, Ti_MJ_rad, coul_log_rad, Q_ie_rad, Q_ie_rad_vol,
    chrd_wt, vert_chord, b_dir, &
 99                                          event_pre, event_post, &
100                                          event_data1, event_data2, event_data3, event_data4,
    event_data5, &
101                                          event_data6, event_data7, event_data8, event_data9,
    event_data10, &
102                                          event_data11, event_data12, event_data13, event_data14,
    event_data15, &
103                                          event_data16, event_data17, event_data18, event_data19,
    event_data20, &
104                                          event_data21, event_data22, event_data23, event_data24,
    event_data25, &
105                                          event_data26, event_data27, event_data28, event_data29,
    event_data30, &
106                                          event_data31, event_data32, event_data33, &
107                                          tri_divert, tri_nondivert, Ti_Te, &
108                                          ni_gof, Ti_gof, Te_gof, ni_19_exp_array, Ti_kev_exp_array,
    Te_kev_exp_array, &
109                                          ni_res_vector, Ti_res_vector, Te_res_vector, dat_temp,
    dat_time_temp
110
111
112 CHARACTER(len=260) :: infile1, infile_global, datafile, datafile_smooth, outfile1, outfile2,
    outfile3, outfile4, outfile5, outfile6, outfile7, outfile8, outfile9, fast_ion_out, &
113                           rad_outfile, outfile_global, outfile_global2, outfile_global3,
    outfile_global4, outfile_global5, outfile_global6, ni_sim_path, cwd, dummy_char, shotnum_string
    (100), shot
114 !CHARACTER(len=6) :: shot
115
116 !Timeseries input stuff
117 real, dimension (:,:), allocatable ::   dat_synced, dat_synced_td, dat_synced_clean,
    dat_synced_smooth, time_array_td, time_array_td_1, time_array_td_2, &
118                                          cleanup_start, cleanup_end, skip_lt, skip_gt, data_td, dat,
    dat_time, dat_smooth
119
120
121 real, dimension (:), allocatable ::     initial_av
122
123 !Main vectors populated with external data
124 real, dimension(:), allocatable ::      t_sim, t_exp, H98, IP, BT0, GASA_CAL, PNBI, POH, Te_dat,
    volume, a_minor, &
125                                          DENsv2, TSTE_TAN, zmaxis, BDOTAMPL, BT0_abs, elong_a, &
126                                          nbi1, nbi2, nbi3, nbi4, nbi5, nbi6, nbi7, nbi8, &
127                                          nbi1e, nbi2e, nbi3e, nbi4e, nbi5e, nbi6e, nbi7e, nbi8e, &
128                                          S_i_nbi1, S_i_nbi2, S_i_nbi3, S_i_nbi4, S_i_nbi5, S_i_nbi6,
    S_i_nbi7, S_i_nbi8, &
129                                          S_i_nbi, S_i_gas ,S_i, S_i_ext_vol, S_i_rec_vol, &
130                                          poh_vol, nbi_vol, nbi_tot, &
131                                          S_i_del_vol1, S_i_del_vol2, S_i_del_vol3, S_i_del_vol4,
    S_i_del_vol5, &
132                                          S_i_del_vol6, S_i_del_vol7, S_i_del_vol8, S_i_del_vol9,
    S_i_del_vol10, S_i_del_vol_tot, S_i_rec_vol_tot,&
133                                          sig_v, tau_e, R0, paux_i, paux_e, nbi_net_frac, coul_log, &
134                                          nz1, nz2, nz3, nz4, nz5, nz6, nz7, nz8, nz9, nz10, z, fz,
    Z_eff, Hz, &
135                                          ni, ne, ni_reduced, ne_reduced, Ti, Te, &
```

```
136                                         wb, delta_wb_i, delta_wb_e, delta_wb_tot, Wb_bin,
    Wb_bin_count, &
137                                         PR_IMP1, PR_IMP2, PR_IMP3, PR_IMP4, PR_IMP5, &
138                                         PR_IMP6, PR_IMP7, PR_IMP8, PR_IMP9, PR_IMP10, PR_IMP_TOT,
    PR_BREM, PRAD, &
139                                         Q_ie, Q_ie1, Q_ie2, Q_ie3, Q_ie_vol, Ti_MJ, Ti_J, Ti_keV,
    Te_MJ, Te_J, Te_keV, ni_19, ne_19, &
140                                         nz1_19, nz2_19, nz3_19, nz4_19, nz5_19, nz6_19, nz7_19,
    nz8_19, nz9_19, nz10_19, &
141                                         P_alpha_e, P_alpha_i, P_nbi_e_vol, P_nbi_i_vol, &
142                                         tau_i_energy, tau_e_energy, tau_i_part, tau_e_part,
    tau_alpha, tau_z, &
143                                         Paux_e_vol, Paux_i_vol, diff_ni_t, diff_ne_t, conf_loss_i,
    conf_loss_e, &
144                                         diff_neTe_t, diff_niTi_t, diff_Te_t, diff_Ti_t, &
145                                         H_i_energy, H_i_energy1, H_i_energy2, H_e_energy, H_i_part,
    H_e_part, H_alpha, H_z, &
146                                         H_i_energy_av, H_e_energy_av, H_i_part_av, H_e_part_av,
    H_alpha_av, H_z_av, &
147                                         taue_raw, tauth_raw, tauth_raw_clean,  taueh98y2_raw, &
148                                         taueh98y2_raw_clean, ptot_raw,ptot_raw_clean, &
149                                         densv2_calc, data_array_td, density_array_td, &
150                                         POH_nTi_V_sumsmooth, &
151                                         te_td, ti_td, te_MJ_td, ti_MJ_td, &
152                                         ne_td, ni_td, ne_19_td, ni_19_td, nz3_td, nz3_19_td, &
153                                         q95_raw, li_raw,  ech_time_on, ich_time_on, &
154                                         tritop_raw,tribot_raw, ni_res, Ti_res, Te_res, xsec_area, &
155                                         nbi_i_frac, nbi_e_frac, fus_loss_i, p_ohm, P_e, P_i,
    pfus_i, &
156                                         pfus_e, coll, rad, conf_e, conf_i, conf_p, q0_raw, &
157                                         nbi_co, nbi_counter, nbi_counter_frac, nbi_short, nbi_long,
    nbi_short_frac, event_change, event_change_rel, ni_0
158   !CHARACTER(len=69) ::   datafile
159   !CHARACTER(len=71) ::   datafile_smooth
160   !CHARACTER(len=72) ::   outfile1, outfile2, outfile3, outfile4, outfile5, outfile6, outfile7,
    outfile8, outfile9, fast_ion_out
161   !CHARACTER(len=73) ::   rad_outfile
162   !CHARACTER(len=65) ::   outfile_global
163   !CHARACTER(len=66) ::   outfile_global2
164   !CHARACTER(len=66) ::   outfile_global3
165   !CHARACTER(len=66) ::   outfile_global4
166   !CHARACTER(len=66) ::   outfile_global5
167   !CHARACTER(len=66) ::   outfile_global6
168   !CHARACTER(len=37) ::   cwd
169   !CHARACTER(len=6)  ::   shotnum_string(100)
170   !CHARACTER(len=3)  ::   ni_sim_path
171   !CHARACTER(len=50)  ::  dummy_char
172   Logical :: file_exists
173
174   Integer, parameter ::   rad_pts=50
175
176   Real, parameter ::      u_0=1.257E-6, e=1.6021E-19, e_reduced = 1.6021, c=2.99793E8,
    eps_0=8.854E-12, eps_0_reduced = 8.854, &
177                                         k_b=1.3804E-23, pi = 3.14159265359, &
178                                         m_d = 2.01410178, m_D_kg = 3.34358348E-27, m_D_kg_reduced =
    3.34358348, &
179                                         m_He_kg_reduced = 6.64465675, m_t = 3.0160492, &
180                                         m_e = 5.48579867E-4, m_e_kg = 9.10938291E-31,
    m_e_kg_reduced = 9.10938291E-4, &
181                                         m_alpha_kg = 6.6446568E-27, m_alpha_kg_reduced = 6.6446568,
    &
182                                         U_alpha_DD_MeV = 0.75, U_alpha_DT_MeV = 3.5, &
183                                         U_alpha_DD_MJ = 1.201575E-19, U_alpha_DT_MJ = 5.60735E-19, &
184                                         R_gas = 62.36367, avogadro = 6.022141E+23, &
185                                         gas_ne_C1 = -0.0059, gas_ne_C2 = 0.3464, &
186                                         gas_Te_C1 = -0.0346, gas_Te_C2 = 4.9726, &
187                                         gas_Ti_C1 = 0.0205, gas_Ti_C2 = 4.1523
188
189   End Module Variables
```

REFERENCES

[1] Weston M Stacey. *Fusion Plasma Physics*. John Wiley & Sons, 2012.

[2] RG Mills. The problem of control of thermonuclear reactors. Technical report, Princeton Univ., NJ Plasma Physics Lab., 1970.

[3] WM Stacey. A survey of thermal instabilities in tokamak plasmas: Theory, comparison with experiment, and predictions for future devices. *Fusion Science and Technology*, 52(1):29, 2007.

[4] C Powell and OJ Hahn. Energy-balance instabilities in fusion plasmas. *Nuclear Fusion*, 12(6):667, 1972.

[5] HP Furth, MN Rosenbluth, PH Rutherford, and W Stodiek. Thermal equilibrium and stability of tokamak discharges. *Physics of Fluids*, 13(12):3020, 1970.

[6] M Ohta, H Yamato, and S Mori. Thermal instability and control of fusion reactor. In *Plasma Physics and Controlled Nuclear Fusion Research 1971. Vol. III. Proceedings of the Fourth International Conference on Plasma Physics and Controlled Nuclear Fusion Research*, 1971.

[7] Ya I Kolesnichenko. Conditions for the quasi-steady development of self-sustaining nuclear reactions. *Nuclear Fusion*, 12(4):419, 1972.

[8] WM Stacey. Operating regimes of controlled thermonuclear reactors and stability against fundamental-mode excursions in particle densities and temperatures. *Nuclear Fusion*, 13(6):843, 1973.

[9] M Shimada, DJ Campbell, V Mukhovatov, M Fujiwara, N Kirneva, K Lackner, M Nagami, VD Pustovitov, N Uckan, J Wesley, et al. Overview and summary. *Nuclear Fusion*, 47(6):S1, 2007.

[10] Weston M Stacey. Principles and rationale of the fusion-fission hybrid burner reactor. In *FUSION FOR NEUTRONS AND SUBCRITICAL NUCLEAR FISSION: Proceedings of the International Conference*, volume 1442, page 31. AIP Publishing, 2012.

[11] WM Stacey, CL Stewart, J-P Floyd, TM Wilks, AP Moore, AT Bopp, MD Hill, S Tandon, and AS Erickson. Resolution of fission and fusion technology integration issues: an upgraded design concept for the subcritical advanced burner reactor. *Nuclear Technology*, 187(1):15, 2014.

[12] WW Pfeiffer, RH Davidson, RL Miller, and RE Waltz. Onetwo: a computer code for modeling plasa transport in tokamaks. Technical report, General Atomic Co., San Diego, CA (USA), 1980.

[13] RJ Hawryluk et al. An empirical approach to tokamak transport. *Physics of plasmas close to thermonuclear conditions*, 1:19, 1980.

[14] RB White and MS Chance. Hamiltonian guiding center drift orbit calculation for plasmas of arbitrary cross section. *Physics of Fluids*, 27(10):2455, 1984.

[15] CR Sovinec, AH Glasser, TA Gianakon, DC Barnes, RA Nebel, SE Kruger, DD Schnack, SJ Plimpton, A Tarditi, MS Chu, et al. Nonlinear magnetohydrodynamics simulation using high-order finite elements. *Journal of Computational Physics*, 195(1):355, 2004.

[16] Weston Monroe Stacey. A coupled plasma-neutrals model for divertor simulations. *Physics of Plasmas*, 5(4):1015, 1998.

[17] Y Ou, TC Luce, E Schuster, JR Ferron, ML Walker, C Xu, and DA Humphreys. Towards model-based current profile control at diii-d. *Fusion Engineering and Design*, 82(5):1153, 2007.

[18] DA Humphreys, JR Ferron, M Bakhtiari, JA Blair, Y In, GL Jackson, H Jhang, RD Johnson, JS Kim, RJ LaHaye, et al. Development of iter-relevant plasma control solutions at diii-d. *Nuclear Fusion*, 47(8):943, 2007.

[19] Yu Gribov, D Humphreys, K Kajiwara, EA Lazarus, JB Lister, Tl Ozeki, A Portone, M Shimada, ACC Sips, and JC Wesley. Plasma operation and control. *Nuclear fusion*, 47(6):S385, 2007.

[20] SC Jardin, N Pomphrey, and J Delucia. Dynamic modeling of transport and positional control of tokamaks. *Journal of computational Physics*, 66(2):481, 1986.

[21] Masanori Murakami, Jin Myung Park, G Giruzzi, J Garcia, P Bonoli, RV Budny, EJ Doyle, A Fukuyama, N Hayashi, M Honda, et al. Integrated modelling of steady-state scenarios and heating and current drive mixes for iter. *Nuclear Fusion*, 51(10):103006, 2011.

[22] Mark D Boyer and Eugenio Schuster. Nonlinear burn control in tokamak fusion reactors via output feedback. In *Proc. 19th World Congress of the International Federation of Automatic Control (Cape Town, South Africa,)*, 2014.

[23] Weigang Hui, Bassam A Bamieh, and George H Miley. Robust burn control of a fusion reactor by modulation of the refueling rate. *Fusion Science and Technology*, 25(3):318, 1994.

[24] Masami Ohnishi, Akira Saiki, and Masao Okamoto. Space-dependent analysis plasma engineering of feedback control to suppress thermal runaway by compression-decompression. *Fusion Science and Technology*, 5(3):326, 1984.

[25] J Mandrekas and WM Stacey. Evaluation of different burn control methods for the international thermonuclear experimental reactor. In *Fusion Engineering, 1989. Proceedings., IEEE Thirteenth Symposium on*, page 404. IEEE, 1989.

[26] Dan Anderson, H Hamnen, M Lisak, T Elevant, and H Persson. Transition to thermonuclear burn in fusion plasmas. *Plasma physics and controlled fusion*, 33(10):1145, 1991.

[27] Scott W Haney, L John Perkins, John Mandrekas, and Weston M Stacey. Active control of burn conditions for the international thermonuclear experimental reactor. *Fusion Science and Technology*, 18(4):606, 1990.

[28] Dan Anderson, Thomas Elevant, Håkan Hamnén, Mietek Lisak, and Hans Persson. Studies of fusion burn control. *Fusion Science and Technology*, 23(1):5, 1993.

[29] LM Hively. Tf-ripple losses from a non-circular tokamak. *Nuclear fusion*, 24(6):779, 1984.

[30] ITER Team et al. Iter physics basis. *Nucl. Fusion*, 39:2137, 1999.

[31] James L Luxon. A design retrospective of the diii-d tokamak. *Nuclear Fusion*, 42(5):614, 2002.

[32] PC Stangeby and GM McCracken. Plasma boundary phenomena in tokamaks. *Nuclear Fusion*, 30(7):1225, 1990.

[33] SL Allen, ME Rensink, DN Hill, DE Perkins, GL Jackson, M Ali Mahdavi, DIII-D Research Team, et al. Recycling and neutral transport in the diii-d tokamak. *Journal of Nuclear Materials*, 162:80, 1989.

[34] Glenn Bateman, Arnold H Kritz, Jon E Kinsey, Aaron J Redd, and Jan Weiland. Predicting temperature and density profiles in tokamaks. *Physics of Plasmas*, 5(5):1793, 1998.

[35] H-S Bosch and GM Hale. Improved formulas for fusion cross-sections and thermal reactivities. *Nuclear Fusion*, 32(4):611, 1992.

[36] Ratko K Janev, William D Langer, Douglas E Post Jr, and Kenneth Evans Jr. Elementary processes in hydrogen-helium plasmas: cross sections and reaction rate coefficients. In *Research supported by DOE,. Berlin and New York, Springer-Verlag (Springer Series on Atoms and Plasmas. Volume 4), 1987, 335 p.*, volume 4, 1987.

[37] J Scharff Lindhard and M Scharff. M. & schiott, he, mat. *Fys. Medd. Dan. Vid. Selsk*, 33:14, 1963.

[38] GM McCracken and PE Stott. Plasma-surface interactions in tokamaks. *Nuclear Fusion*, 19(7):889, 1979.

[39] Gianfranco Federici, Charles H Skinner, Jeffrey N Brooks, Joseph Paul Coad, Christian Grisolia, Anthony A Haasz, Ahmed Hassanein, Volker Philipps, C Spencer Pitcher, Joachim Roth, et al. Plasma-material interactions in current tokamaks and their implications for next step fusion reactors. *Nuclear Fusion*, 41(12):1967, 2001.